RM0083
Reference manual

32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications

Introduction

The SPC56xP60/54 microcontroller is built on the Power Architecture® platform and targets chassis market segment, specifically the airbag application space. The Power Architecture based 32-bit microcontrollers represent the latest achievement in integrated automotive application controllers.

SPC56xP60/54 devices are built around a dual-core platform for optimized performance-power consumption trade-off but is available also as single-core version.

Contents

Preface 52
Overview 52
Audience 52
Chapter organization and device-specific information 52
References 52

1 Introduction 53
1.1 The SPC56xP60/54 microcontroller family 53
1.2 Application examples 53
1.3 SPC56xP60/54 Device Summary 54
1.4 Device block diagram 56
1.5 Critical performance parameters 60
1.6 Feature summary 61
1.7 Features details 62
1.7.1 High performance e200z0h core processor 62
1.7.2 Crossbar switch (XBAR) 62
1.7.3 Enhanced direct memory access (eDMA) 63
1.7.4 On-chip flash memory with ECC 63
1.7.5 On-chip SRAM with ECC 64
1.7.6 Interrupt controller (INTC) 64
1.7.7 System clocks and clock generation 65
1.7.8 Frequency modulated phase-locked loop (FMPLL) 65
1.7.9 Main oscillator 66
1.7.10 Internal RC oscillator 66
1.7.11 Periodic interrupt timer (PIT) 66
1.7.12 System timer module (STM) 66
1.7.13 Software watchdog timer (SWT) 67
1.7.14 Fault collection and control unit (FCCU) 67
1.7.15 System integration unit (SIUL) 67
1.7.16 Boot and censorship 67
1.7.17 Error correction status module (ECSM) 68
1.7.18 FlexCAN 68
1.7.19 Safety port (FlexCAN) 69

1.7.20 FlexRay . . . 70
1.7.21 Serial communication interface module (LINFlex) . . . 70
1.7.22 Deserial serial peripheral interface (DSPI) . . . 71
1.7.23 eTimer . . . 71
1.7.24 Analog-to-digital converter (ADC) . . . 72
1.7.25 Cross triggering unit (CTU) . . . 72
1.7.26 Cyclic redundancy check (CRC) . . . 73
1.7.27 Nexus development interface (NDI) . . . 73
1.7.28 IEEE 1149.1 (JTAG) controller . . . 74
1.7.29 On-chip voltage regulator (VREG) . . . 75
1.8 Developer environment . . . 75

2 Memory Map . . . 76

3 Signal Description . . . 79
3.1 176-pin LQFP pinout . . . 79
3.2 144-pin LQFP pinout . . . 80
3.3 100-pin LQFP pinout . . . 81
3.4 Pin descriptions . . . 82
3.4.1 Power supply and reference voltage pins . . . 82
3.4.2 System pins . . . 83
3.4.3 Pin muxing . . . 85
3.5 CTU / ADCs / eTimers / DSPI / FlexRay connections . . . 100

4 Clock Description . . . 102
4.1 Clock architecture . . . 102
4.2 Available clock domains . . . 106
4.2.1 FMPLL input reference clock . . . 106
4.2.2 Clock selectors . . . 106
4.2.3 Auxiliary Clock Selector 0 . . . 106
4.2.4 Auxiliary Clock Selector 1 . . . 106
4.2.5 Auxiliary Clock Selector 2 . . . 106
4.2.6 Auxiliary clock dividers . . . 107
4.2.7 External clock divider . . . 107
4.3 Alternate module clock domains . . . 108
4.3.1 FlexCAN clock domains . . . 108

4.3.2 FlexRay clock domains . . . 108
4.3.3 SWT clock domains . . . 108
4.3.4 Nexus Message Clock (MCKO) . . . 108
4.3.5 Cross Triggering Unit (CTU) clock domains . . . 108
4.3.6 Peripherals . . . 109
4.4 Clock behavior in STOP and HALT mode . . . 109
4.5 Software controlled power management/clock gating . . . 110
4.6 System clock functional safety . . . 111
4.7 IRC 16 MHz internal RC oscillator (RC_CTL) . . . 111
4.8 XOSC external crystal oscillator . . . 112
4.8.1 Functional description . . . 112
4.8.2 Register description . . . 112
4.9 Frequency Modulated Phase Locked Loop (FMPLL) . . . 113
4.9.1 Introduction . . . 113
4.9.2 Overview . . . 114
4.9.3 Features . . . 114
4.9.4 Memory map . . . 114
4.9.5 Register description . . . 115
4.9.6 Functional description . . . 118
4.9.7 Recommendations . . . 121
4.10 Clock Monitor Unit (CMU) . . . 122
4.10.1 Overview . . . 122
4.10.2 Main features . . . 123
4.10.3 Functional description . . . 123
4.10.4 Memory map and register description . . . 125

5 Clock Generation Module (MC_CGM) . . . 133
5.1 Introduction . . . 133
5.1.1 Overview . . . 133
5.1.2 Features . . . 135
5.2 External Signal Description . . . 135
5.3 Memory Map and Register Definition . . . 135
5.3.1 Register Descriptions . . . 139
5.4 Functional Description . . . 147
5.4.1 System Clock Generation . . . 147
5.4.2 Auxiliary Clock Generation . . . 148

5.4.3 Dividers Functional Description 151
5.4.4 Output Clock Multiplexing 151
5.4.5 Output Clock Division Selection 152

6 Mode Entry Module (MC_ME) 153
6.1 Introduction 153
6.1.1 Overview 153
6.1.2 Features 155
6.1.3 Modes of Operation 155
6.2 External Signal Description 156
6.3 Memory Map and Register Definition 156
6.3.1 Memory Map 156
6.3.2 Register Description 164
6.4 Functional Description 186
6.4.1 Mode Transition Request 186
6.4.2 Modes Details 187
6.4.3 Mode Transition Process 190
6.4.4 Protection of Mode Configuration Registers 198
6.4.5 Mode Transition Interrupts 198
6.4.6 Peripheral Clock Gating 200
6.4.7 Application Example 200

7 Power Control Unit (MC_PCU) 202
7.1 Introduction 202
7.1.1 Overview 202
7.1.2 Features 202
7.2 External Signal Description 203
7.3 Memory Map and Register Definition 203
7.3.1 Memory Map 203
7.3.2 Register Descriptions 204

8 Reset Generation Module (MC_RGM). 207
8.1 Introduction 207
8.1.1 Overview 207
8.1.2 Features 209
8.1.3 Reset Sources 209

8.2 External Signal Description 210
8.3 Memory Map and Register Definition 210
8.3.1 Register Descriptions 212
8.4 Functional Description 222
8.4.1 Reset State Machine 222
8.4.2 Destructive Resets 224
8.4.3 External Reset 225
8.4.4 Functional Resets 225
8.4.5 Alternate Event Generation 226
8.4.6 Boot Mode Capturing 226

9 Interrupt Controller (INTC). 228
9.1 Introduction 228
9.2 Features 228
9.3 Block diagram 230
9.4 Modes of operation 230
9.4.1 Normal mode 230
9.5 Memory map and registers description 232
9.5.1 Module memory map 232
9.5.2 Registers description 232
9.6 Functional description 241
9.6.1 Interrupt request sources 250
9.6.2 Priority management 251
9.6.3 Handshaking with processor 252
9.7 Initialization/application information 254
9.7.1 Initialization flow 254
9.7.2 Interrupt exception handler 255
9.7.3 ISR, RTOS, and task hierarchy 257
9.7.4 Order of execution 257
9.7.5 Priority ceiling protocol 258
9.7.6 Selecting priorities according to request rates and deadlines 259
9.7.7 Software configurable interrupt requests 259
9.7.8 Lowering priority within an ISR 260
9.7.9 Negating an interrupt request outside of its ISR 261
9.7.10 Examining LIFO contents 261

10 System Status and Configuration Module (SSCM) 262
10.1 Introduction 262
10.1.1 Overview 262
10.1.2 Features 262
10.1.3 Modes of operation 263
10.2 Memory map and register description 263
10.2.1 Memory map 263
10.2.2 Register description 263
10.3 Functional description 272
10.4 Initialization/application information 272
10.4.1 Reset 272

11 System Integration Unit Lite (SIUL) 273
11.1 Introduction 273
11.2 Overview 273
11.3 Features 275
11.3.1 Register protection 275
11.4 External signal description 275
11.4.1 Detailed signal descriptions 276
11.5 Memory map and register description 276
11.5.1 SIUL memory map 276
11.5.2 Register description 277
11.6 Functional description 293
11.6.1 General 293
11.6.2 Pad control 293
11.6.3 General purpose input and output pads (GPIO) 293
11.6.4 External interrupts 294
11.7 Pin muxing 295

12 e200z0 and e200z0h Core 296
12.1 Overview 296
12.2 Features 296
12.2.1 Microarchitecture summary 297
12.3 Core registers and programmer’s model 301
12.3.1 Unimplemented SPRs and read-only SPRs 304

12.4 Instruction summary 304

13 Peripheral Bridge (PBRIDGE) 305

13.1 Introduction 305
13.2 Block interface 305
13.3 Features 306
13.4 Memory map and register description 306
13.4.1 Register access 306
13.4.2 Memory map 306
13.4.3 Register descriptions 307
13.5 Functional description 311
13.5.1 Access support 311
13.5.2 General operation 311

14 Crossbar Switch (XBAR) 313

14.1 Information specific to this device 313
14.1.1 Register availability 313
14.1.2 MPR reset value 313
14.1.3 max_halt_request signal value 313
14.1.4 Logical master IDs 313
14.1.5 Master port allocation 313
14.1.6 Slave port allocation 314
14.2 Introduction 314
14.2.1 Overview 314
14.2.2 Features 315
14.2.3 Limitations 315
14.2.4 General Operation 315
14.3 XBAR registers 316
14.3.1 Register summary 316
14.3.2 XBAR register descriptions 318
14.3.3 Coherency 325
14.4 Function 325
14.4.1 Arbitration 325
14.4.2 Priority Assignment 327
14.4.3 Master Port Functionality 327
14.4.4 Slave Port Functionality 329

14.5 Initialization/Application Information 336
14.6 Interface 336
14.6.1 Overview 336
14.6.2 Master Ports 336
14.6.3 Slave Ports 337

15 Error Correction Status Module (ECSM) 339
15.1 Introduction 339
15.2 Overview 339
15.3 Features 339
15.4 Memory map and registers description 339
15.4.1 Memory map 339
15.4.2 Registers description 341

16 Internal Static RAM (SRAM). 360
16.1 Introduction 360
16.2 SRAM operating mode 360
16.3 Module memory map 360
16.4 Register descriptions 360
16.5 SRAM ECC mechanism 360
16.5.1 Access timing 361
16.5.2 Reset effects on SRAM accesses 362
16.6 Functional description 362
16.7 Initialization and application information 362

17 Flash Memory 363
17.1 Introduction 363
17.2 Platform Flash controller 365
17.2.1 Introduction 365
17.2.2 Modes of operation 366
17.2.3 External signal descriptions 366
17.2.4 Memory map and registers description 366
17.2.5 Functional description 368
17.2.6 Basic interface protocol 369
17.2.7 Access protections 369
17.2.8 Read cycles — buffer miss 370

17.2.9 Read cycles — buffer hit . . . 370
17.2.10 Write cycles . . . 370
17.2.11 Error termination . . . 370
17.2.12 Access pipelining . . . 371
17.2.13 Flash error response operation . . . 371
17.2.14 Bank0 page read buffers and prefetch operation . . . 371
17.2.15 Bank1 temporary holding register . . . 374
17.2.16 Read-While-Write functionality . . . 374
17.2.17 Wait state emulation . . . 376
17.2.18 Timing diagrams . . . 377
17.3 Flash memory . . . 384
17.3.1 Introduction . . . 384
17.3.2 Main features . . . 384
17.3.3 Block diagram . . . 384
17.3.4 Functional description . . . 386
17.3.5 Operating modes . . . 390
17.3.6 Registers description . . . 393
17.3.7 Register map . . . 393
17.3.8 Code Flash programming considerations . . . 431

18 Memory Protection Unit (MPU) . . . 443
18.1 Introduction . . . 443
18.2 Block diagram . . . 443
18.3 Features . . . 445
18.4 Modes of operation . . . 445
18.5 External signal description . . . 446
18.6 Memory map and register definition . . . 446
18.6.1 MPU Control/Error Status Register (MPU_CESR) . . . 448
18.6.2 MPU Error Address Register, Slave Port n (MPU_EARn) . . . 448
18.6.3 MPU Error Detail Register, Slave Port n (MPU_EDRn) . . . 449
18.6.4 MPU Region Descriptor n (MPU_RGDn) . . . 450
18.6.5 MPU Region Descriptor Alternate Access Control n (MPU_RGDAACn) . . . 454
18.7 Functional description . . . 456
18.7.1 Access evaluation macro . . . 456
18.7.2 Putting it all together and AHB error terminations . . . 459

18.8 Initialization information 460
18.9 Application information 460

19 Enhanced Direct Memory Access (eDMA) 463
19.1 Introduction 463
19.2 Overview 463
19.3 Features 464
19.4 Modes of operation 464
19.4.1 Normal mode 464
19.4.2 Debug mode 465
19.5 Memory map and register definition 465
19.5.1 Memory map 465
19.5.2 Register descriptions 467
19.6 Functional description 487
19.6.1 eDMA microarchitecture 487
19.6.2 eDMA basic data flow 488
19.6.3 eDMA performance 491
19.7 Initialization / application information 495
19.7.1 eDMA initialization 495
19.7.2 DMA programming errors 497
19.7.3 DMA request assignments 497
19.7.4 DMA arbitration mode considerations 498
19.7.5 DMA transfer 498
19.7.6 TCD status 502
19.7.7 Channel linking 503
19.7.8 Dynamic programming 504

20 DMA Channel Mux (DMA_MUX) 505
20.1 Introduction 505
20.1.1 Overview 505
20.1.2 Features 505
20.1.3 Modes of operation 506
20.2 External signal description 506
20.2.1 Overview 506
20.3 Memory map and register definition 506
20.3.1 Memory map 506

20.3.2 Register descriptions 508
20.4 DMA request mapping 509
20.5 Functional description 510
20.5.1 DMA channels with periodic triggering capability 510
20.5.2 DMA channels with no triggering capability 513
20.6 Initialization/application information 513
20.6.1 Reset 513
20.6.2 Enabling and configuring sources 513

21 FlexRay Communication Controller (FlexRay) 518
21.1 Introduction 518
21.1.1 Color coding 518
21.1.2 Overview 518
21.1.3 Features 520
21.1.4 Modes of operation 521
21.2 External signal description 521
21.2.1 Detailed signal descriptions 522
21.3 Controller host interface clocking 523
21.4 Protocol engine clocking 523
21.4.1 PLL Clocking 523
21.4.2 Oscillator Clocking 523
21.5 Memory map and register description 523
21.5.1 Memory map 523
21.5.2 Register descriptions 526
21.6 Functional description 596
21.6.1 Message buffer concept 596
21.6.2 Physical message buffer 596
21.6.3 Message buffer types 597
21.6.4 FlexRay memory layout 602
21.6.5 Physical message buffer description 604
21.6.6 Individual message buffer functional description 614
21.6.7 Individual message buffer search 639
21.6.8 Individual message buffer reconfiguration 642
21.6.9 Receive FIFO 643
21.6.10 Channel device modes 647
21.6.11 External clock synchronization 649

21.6.12 Sync frame ID and sync frame deviation tables 650
21.6.13 MTS generation 653
21.6.14 Key slot transmission 654
21.6.15 Sync frame filtering 654
21.6.16 Strobe signal support 655
21.6.17 Timer support 656
21.6.18 Slot status monitoring 657
21.6.19 System bus access 660
21.6.20 Interrupt support 661
21.6.21 Lower bit rate support 664
21.7 Application information 665
21.7.1 Initialization sequence 665
21.7.2 Shut down sequence 666
21.7.3 Number of usable message buffers 666
21.7.4 Protocol control command execution 667
21.7.5 Protocol reset command 668
21.7.6 Message buffer search on simple message buffer configuration 668

22 Deserial Serial Peripheral Interface (DSPI) 672
22.1 Introduction 672
22.2 Block diagram 672
22.3 Overview 673
22.4 Features 673
22.5 Modes of operation 674
22.5.1 Master mode 675
22.5.2 Slave mode 675
22.5.3 Module disable mode 675
22.5.4 Debug mode 675
22.6 External signal description 675
22.6.1 Signal overview 675
22.6.2 Signal names and descriptions 676
22.7 Memory map and registers description 677
22.7.1 Memory map 677
22.7.2 Registers description 678
22.8 Functional description 695
22.8.1 Modes of operation 696

22.8.2 Start and stop of DSPI transfers . . . 697
22.8.3 Serial Peripheral Interface (SPI) configuration . . . 698
22.8.4 DSPI baud rate and clock delay generation . . . 701
22.8.5 Transfer formats . . . 704
22.8.6 Continuous Serial communications clock . . . 711
22.8.7 Interrupts/DMA requests . . . 713
22.8.8 Power saving features . . . 714
22.9 Initialization and application information . . . 715
22.9.1 Managing queues . . . 715
22.9.2 Baud rate settings . . . 715
22.9.3 Delay settings . . . 717
22.9.4 Calculation of FIFO pointer addresses . . . 717

23 LIN Controller (LINFlex) . . . 720
23.1 Introduction . . . 720
23.2 Main features . . . 720
23.2.1 LIN mode features . . . 720
23.2.2 UART mode features . . . 720
23.2.3 Features common to LIN and UART . . . 721
23.3 General description . . . 721
23.4 Fractional baud rate generation . . . 722
23.5 Operating modes . . . 724
23.5.1 Initialization mode . . . 724
23.5.2 Normal mode . . . 724
23.5.3 Low power mode (Sleep) . . . 724
23.6 Test modes . . . 725
23.6.1 Loop Back mode . . . 725
23.6.2 Self Test mode . . . 725
23.7 Memory map and registers description . . . 726
23.7.1 Memory map . . . 726
23.8 Functional description . . . 753
23.8.1 UART mode . . . 753
23.8.2 LIN mode . . . 755
23.8.3 8-bit timeout counter . . . 763
23.8.4 Interrupts . . . 764

24 FlexCAN 766
24.1 Introduction 766
24.1.1 Overview 766
24.1.2 FlexCAN module features 767
24.1.3 Modes of operation 768
24.2 External signal description 769
24.2.1 Overview 769
24.2.2 Signal Descriptions 769
24.3 Memory map and registers description 769
24.3.1 FlexCAN memory mapping 769
24.3.2 Message buffer structure 771
24.3.3 Rx FIFO structure 775
24.3.4 Registers description 777
24.4 Functional description 795
24.4.1 Overview 795
24.4.2 Transmit process 795
24.4.3 Arbitration process 796
24.4.4 Receive process 796
24.4.5 Matching process 798
24.4.6 Data coherence 799
24.4.7 Rx FIFO 801
24.4.8 CAN protocol related features 802
24.4.9 Modes of operation details 806
24.4.10 Interrupts 807
24.4.11 Bus interface 808
24.5 Initialization/application information 808
24.5.1 FlexCAN initialization sequence 809

25 Analog-to-Digital Converter (ADC) 810
25.1 Overview 810
25.1.1 Device-specific features 810
25.1.2 Device-specific pin configuration features 810
25.1.3 Device-specific implementation 811
25.2 Introduction 811
25.3 Functional description 812
25.3.1 Analog channel conversion 812

25.3.2 Analog clock generator and conversion timings . . . 815
25.3.3 ADC sampling and conversion timing . . . 816
25.3.4 ADC CTU (Cross Triggering Unit) . . . 818
25.3.5 Presampling . . . 819
25.3.6 Programmable analog watchdog . . . 820
25.3.7 DMA functionality . . . 821
25.3.8 Interrupts . . . 821
25.3.9 Power-down mode . . . 822
25.3.10 Auto-clock-off mode . . . 822
25.4 Register descriptions . . . 822
25.4.1 Introduction . . . 822
25.4.2 Control logic registers . . . 824
25.4.3 Interrupt registers . . . 828
25.4.4 DMA registers . . . 832
25.4.5 Threshold registers . . . 834
25.4.6 Presampling registers . . . 835
25.4.7 Conversion Timing Registers CTR[0] . . . 837
25.4.8 Mask registers . . . 837
25.4.9 Delay registers . . . 839
25.4.10 Data registers . . . 839

26 Cross Triggering Unit (CTU) . . . 841
26.1 Introduction . . . 841
26.2 CTU overview . . . 841
26.3 Functional description . . . 842
26.3.1 Trigger events features . . . 842
26.3.2 Trigger generator subunit (TGS) . . . 843
26.3.3 TGS in triggered mode . . . 843
26.3.4 TGS in sequential mode . . . 844
26.3.5 TGS counter . . . 845
26.4 Scheduler subunit (SU) . . . 846
26.4.1 ADC commands list . . . 848
26.4.2 ADC commands list format . . . 848
26.4.3 ADC results . . . 850
26.5 Reload mechanism . . . 850
26.6 Power safety mode . . . 851

26.6.1 MDIS bit . . . 852
26.6.2 STOP mode . . . 852
26.7 Interrupts and DMA requests . . . 852
26.7.1 DMA support . . . 852
26.7.2 CTU faults and errors . . . 852
26.7.3 CTU interrupt/DMA requests . . . 853
26.8 Memory map . . . 855
26.8.1 Trigger Generator Sub-unit Input Selection Register (TGSISR) . . . 859
26.8.2 Trigger Generator Sub-unit Control Register (TGSCR) . . . 861
26.8.3 Trigger x Compare Register (TxCR, *x* = 0...7) . . . 862
26.8.4 TGS Counter Compare Register (TGSCCR) . . . 862
26.8.5 TGS Counter Reload Register (TGSCRR) . . . 863
26.8.6 Commands list control register 1 (CLCR1) . . . 863
26.8.7 Commands list control register 2 (CLCR2) . . . 864
26.8.8 Trigger handler control register 1 (THCR1) . . . 864
26.8.9 Trigger handler control register 2 (THCR2) . . . 866
26.8.10 Commands list register x (x = 1,...,24) (CLR*x*) . . . 868
26.8.11 FIFO DMA control register (FDCR) . . . 869
26.8.12 FIFO control register (FCR) . . . 870
26.8.13 FIFO threshold register (FTH) . . . 872
26.8.14 FIFO status register (FST) . . . 872
26.8.15 FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*) . . . 874
26.8.16 FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*) . . . 875
26.8.17 Cross triggering unit error flag register (CTUEFR) . . . 875
26.8.18 Cross triggering unit interrupt flag register (CTUIFR) . . . 876
26.8.19 Cross triggering unit interrupt/DMA register (CTUIR) . . . 877
26.8.20 Control ON time register (COTR) . . . 878
26.8.21 Cross triggering unit control register (CTUCR) . . . 879
26.8.22 Cross triggering unit digital filter (CTUDF) . . . 880
26.8.23 Cross triggering unit power control register (CTUPCR) . . . 880

27 Semaphore Unit (SEMA4) . . . 881
27.1 Introduction . . . 881
27.1.1 Block diagram . . . 881
27.1.2 Features . . . 883
27.1.3 Modes of operation . . . 883
27.2 Signal description . . . 883

27.3 Memory map and register description . . . 883
27.3.1 Semaphores gate *n* register (SEMA4_GATE*n*) . . . 884
27.3.2 Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE) . 885
27.3.3 Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF) . . . 886
27.3.4 Semaphores (secure) reset gate *n* (SEMA4_RSTGT) . . . 886
27.3.5 Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF) . . . 888
27.4 Functional description . . . 890
27.4.1 Semaphore usage . . . 891
27.5 Initialization information . . . 892
27.6 Application information . . . 892
27.7 DMA requests . . . 893

28 eTimer . . . 894
28.1 Introduction . . . 894
28.2 Features . . . 895
28.3 Module block diagram . . . 896
28.4 Channel block diagram . . . 897
28.5 External signal descriptions . . . 897
28.5.1 ETC[5:0]—eTimer input/outputs . . . 897
28.6 Memory map and registers . . . 897
28.6.1 Overview . . . 897
28.6.2 Timer channel registers . . . 901
28.6.3 Watchdog timer registers . . . 916
28.6.4 Configuration registers . . . 917
28.7 Functional description . . . 920
28.7.1 General . . . 920
28.7.2 Counting modes . . . 920
28.7.3 Other features . . . 924
28.8 Clocks . . . 925
28.9 Interrupts . . . 926
28.10 DMA . . . 927

29 Functional Safety . . . 928
29.1 Introduction . . . 928
29.2 Register protection module . . . 928

29.2.1 Overview . . . 928
29.2.2 Features . . . 929
29.2.3 Modes of operation . . . 929
29.2.4 External signal description . . . 929
29.2.5 Memory map and registers description . . . 929
29.2.6 Functional description . . . 933
29.2.7 Reset . . . 936
29.3 Software Watchdog Timer (SWT) . . . 936
29.3.1 Overview . . . 936
29.3.2 Features . . . 937
29.3.3 Modes of operation . . . 937
29.3.4 External signal description . . . 937
29.3.5 SWT memory map and registers description . . . 937
29.3.6 Functional description . . . 943

30 Fault Collection and Control Unit (FCCU) . . . 946
30.1 Introduction . . . 946
30.1.1 Glossary and acronyms . . . 946
30.2 Main features . . . 947
30.3 Block diagram . . . 947
30.4 Signal description . . . 948
30.5 Register interface . . . 949
30.6 Memory map and register description . . . 949
30.6.1 FCCU Control Register (FCCU_CTRL) . . . 951
30.6.2 FCCU CTRL Key Register (FCCU_CTRLK) . . . 954
30.6.3 FCCU Configuration Register (FCCU_CFG) . . . 955
30.6.4 FCCU CF Configuration Register (FCCU_CF_CFG0..3) . . . 957
30.6.5 FCCU NCF Configuration Register (FCCU_NCF_CFG0..3) . . . 958
30.6.6 FCCU CFS Configuration Register (FCCU_CFS_CFG0..7) . . . 959
30.6.7 FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7) . . . 961
30.6.8 FCCU CF Status Register (FCCU_CFS0..3) . . . 961
30.6.9 FCCU CF Key Register (FCCU_CFK) . . . 963
30.6.10 FCCU NCF Status Register (FCCU_NCFS0..3) . . . 963
30.6.11 FCCU NCF Key Register (FCCU_NCFK) . . . 965
30.6.12 FCCU NCF Enable Register (FCCU_NCFE0..3) . . . 966
30.6.13 FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3) . . . 967

30.6.14 FCCU NCF Time-out Register (FCCU_NCF_TO) . . . 967
30.6.15 FCCU CFG Timeout Register (FCCU_CFG_TO) . . . 968
30.6.16 FCCU Status Register (FCCU_STAT) . . . 969
30.6.17 FCCU CF Fake Register (FCCU_CFF) . . . 971
30.6.18 FCCU NCF Fake Register (FCCU_NCFF) . . . 972
30.6.19 FCCU IRQ Status Register (FCCU_IRQ_STAT) . . . 972
30.6.20 FCCU IRQ Enable Register (FCCU_IRQ_EN) . . . 973
30.6.21 FCCU XTMR Register (FCCU_XTMR) . . . 974
30.6.22 FCCU MCS Register (FCCU_MCS) . . . 975
30.7 Functional description . . . 976
30.7.1 Definitions . . . 976
30.7.2 FSM description . . . 976
30.7.3 Self-checking capabilities . . . 978
30.7.4 Reset interface . . . 979
30.7.5 Fault priority scheme and nesting . . . 979
30.7.6 Fault recovery . . . 980
30.7.7 NVM interface . . . 984
30.7.8 FCCU_F interface . . . 985
30.7.9 Fault mapping . . . 989

31 Wakeup Unit (WKPU) . . . 991
31.1 Overview . . . 991
31.2 Features . . . 991
31.3 External signal description . . . 991
31.4 Memory map and registers description . . . 991
31.4.1 Memory map . . . 991
31.4.2 Registers description . . . 992
31.5 Functional description . . . 994
31.5.1 General . . . 994
31.5.2 Non-Maskable Interrupts . . . 994

32 Periodic Interrupt Timer (PIT) . . . 996
32.1 Introduction . . . 996
32.1.1 Overview . . . 996
32.1.2 Features . . . 996
32.2 Signal description . . . 997

32.3 Memory map and registers description 997
32.3.1 Memory map 997
32.3.2 Registers description 998
32.4 Functional description 1002
32.4.1 General 1002
32.4.2 Interrupts 1004
32.5 Initialization and application information 1004
32.5.1 Example configuration 1004

33 System Timer Module (STM) 1005
33.1 Overview 1005
33.2 Features 1005
33.3 Modes of operation 1005
33.4 External signal description 1005
33.5 Memory map and registers description 1005
33.5.1 Memory map 1005
33.5.2 Registers description 1006
33.6 Functional description 1010

34 Cyclic Redundancy Check (CRC) 1011
34.1 Introduction 1011
34.1.1 Glossary 1011
34.2 Main features 1011
34.2.1 Standard features 1011
34.3 Block diagram 1011
34.3.1 IPS bus interface 1012
34.4 Functional description 1012
34.5 Memory map and registers description 1014
34.5.1 CRC Configuration Register (CRC_CFG) 1015
34.5.2 CRC Input Register (CRC_INP) 1016
34.5.3 CRC Current Status Register (CRC_CSTAT) 1017
34.5.4 CRC Output Register (CRC_OUTP) 1018
34.5.5 CRC Output Check Register (CRC_OUTP_CHK) 1018
34.6 Use cases and limitations 1019

35 Boot Assist Module (BAM) 1023

35.1 Overview . . . 1023
35.2 Features . . . 1023
35.3 Boot modes . . . 1023
35.4 Memory map . . . 1023
35.5 Functional description . . . 1024
35.5.1 Entering boot modes . . . 1024
35.5.2 SPC56xP60/54 boot pins . . . 1025
35.5.3 Reset Configuration Half Word (RCHW) . . . 1026
35.5.4 Single chip boot mode . . . 1027
35.5.5 Boot through BAM . . . 1028
35.5.6 Boot from UART—autobaud disabled . . . 1034
35.5.7 Bootstrap with FlexCAN—autobaud disabled . . . 1035
35.6 FlexCAN boot mode download protocol . . . 1036
35.6.1 Autobaud feature . . . 1036
35.6.2 Interrupt . . . 1048
35.7 Censorship . . . 1048

36 Voltage Regulators and Power Supplies . . . 1053
36.1 Voltage regulator . . . 1053
36.1.1 High Power or Main Regulator (HPREG) . . . 1053
36.1.2 Low Voltage Detectors (LVD) and Power On Reset (POR) . . . 1053
36.1.3 VREG digital interface . . . 1054
36.1.4 Registers Description . . . 1055
36.2 Power supply strategy . . . 1056

37 IEEE 1149.1 Test Access Port Controller (JTAGC) . . . 1058
37.1 Introduction . . . 1058
37.2 Block diagram . . . 1058
37.3 Overview . . . 1058
37.4 Features . . . 1059
37.5 Modes of operation . . . 1059
37.5.1 Reset . . . 1059
37.5.2 IEEE 1149.1-2001 defined test modes . . . 1059
37.6 External signal description . . . 1060
37.7 Memory map and registers description . . . 1060

37.7.1 Instruction register 1061
37.7.2 Bypass register 1061
37.7.3 Device identification register 1061
37.7.4 Boundary scan register 1062
37.8 Functional description 1062
37.8.1 JTAGC reset configuration 1062
37.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port (TAP) 1062
37.8.3 TAP controller state machine 1063
37.8.4 JTAGC instructions 1065
37.8.5 Boundary scan 1067
37.9 e200z0 OnCE controller 1067
37.9.1 e200z0 OnCE controller block diagram 1068
37.9.2 e200z0 OnCE controller functional description 1068
37.9.3 e200z0 OnCE controller registers description 1068
37.10 Initialization/Application Information 1070

38 Nexus Port Controller (NPC) 1071
38.1 Information specific to this device 1071
38.1.1 Parameter values 1071
38.1.2 Unavailable features 1071
38.2 Introduction 1071
38.2.1 Overview 1072
38.2.2 Features 1072
38.2.3 Modes of operation 1073
38.3 External signal description 1074
38.3.1 Overview 1074
38.3.2 Detailed signal descriptions 1074
38.4 Register definition 1075
38.4.1 Register descriptions 1076
38.5 Functional description 1080
38.5.1 NPC reset configuration 1080
38.5.2 Auxiliary Output Port 1080
38.5.3 IEEE 1149.1-2001 (JTAG) TAP 1083
38.5.4 Nexus JTAG Port Sharing 1088
38.5.5 MCKO and ipg_sync_mcko 1088
38.5.6 EVTO Sharing 1088

38.5.7 Nexus Reset Control 1088
38.5.8 System Clock Locked Indication 1089
38.6 Initialization/Application Information 1089
38.6.1 Accessing NPC tool-mapped registers 1089

Appendix A Registers Under Protection 1090

Revision History 1103

List of tables

Table 1. SPC56xP60/54 device comparison . . . 54
Table 2. SPC56xP60/54 device configuration difference . . . 56
Table 3. SPC56xP60/54 series block summary . . . 58
Table 4. Memory map . . . 76
Table 5. Supply pins . . . 82
Table 6. System pins . . . 84
Table 7. Pin muxing . . . 85
Table 8. CTU / ADCs / eTimers / DSPI / FlexRay connections . . . 100
Table 9. Output Clock Division 256 Select Register (CLK_DIV_256) Field Descriptions . . . 107
Table 10. Software controlled power management/clock gating support . . . 110
Table 11. RC_CTL field descriptions . . . 111
Table 12. Crystal oscillator truth table . . . 112
Table 13. OSC_CTL memory map . . . 112
Table 14. OSC_CTL field descriptions . . . 113
Table 15. FMPLL memory map . . . 115
Table 16. CR field descriptions . . . 116
Table 17. MR field descriptions . . . 117
Table 18. Progressive clock switching on pll_select rising edge . . . 119
Table 19. CMU module summary . . . 122
Table 20. CMU memory map . . . 125
Table 21. CMU_0_CSR field descriptions . . . 126
Table 22. CMU_0_FDR field descriptions . . . 127
Table 23. CMU_0_HFREFR_0 field descriptions . . . 127
Table 24. CMU_0_LFREFR_0 fields descriptions . . . 128
Table 25. CMU_0_ISR field descriptions . . . 128
Table 26. CMU_0_MDR field descriptions . . . 129
Table 27. CMU_1_CSR field descriptions . . . 130
Table 28. CMU_1_HFREFR_0 field descriptions . . . 130
Table 29. CMU_1_LFREFR_0 field descriptions . . . 131
Table 30. CMU_1_ISR field descriptions . . . 131
Table 31. MC_CGM Register Description . . . 135
Table 32. MC_CGM Memory Map . . . 136
Table 33. Output Clock Enable Register (CGM_OC_EN) Field Descriptions . . . 139
Table 34. Output Clock Enable Register (CGM_OC_EN) Field Descriptions . . . 140
Table 35. System Clock Select Status Register (CGM_SC_SS) Field Descriptions . . . 141
Table 36. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) Field Descriptions . . . 142
Table 37. Auxiliary Clock 0 Divider 0 Configuration Register (CGM_AC0_DC0) Field Descriptions 143
Table 38. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) Field Descriptions . . . 144
Table 39. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0) Field Descriptions 144
Table 40. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) Field Descriptions . . . 145
Table 41. Auxiliary Clock 2 Divider 0 Configuration Register (CGM_AC2_DC0) Field Descriptions 146
Table 42. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) Field Descriptions . . . 147
Table 43. Auxiliary Clock 3 Divider 0 Configuration Register (CGM_AC3_DC0) Field Descriptions 147
Table 44. MC_ME Mode Descriptions . . . 155
Table 45. MC_ME Register Description . . . 156
Table 46. MC_ME Memory Map . . . 160
Table 47. Global Status Register (ME_GS) Field Descriptions . . . 165
Table 48. Mode Control Register (ME_MCTL) Field Descriptions . . . 168

Table 49. Mode Enable Register (ME_ME) Field Descriptions. . . . 169
Table 50. Interrupt Status Register (ME_IS) Field Descriptions . . . 170
Table 51. Interrupt Mask Register (ME_IM) Field Descriptions . . . 171
Table 52. Invalid Mode Transition Status Register (ME_IMTS) Field Descriptions . . . 172
Table 53. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions . . . 174
Table 54. Mode Configuration Registers (ME_<mode>_MC) Field Descriptions . . . 180
Table 55. Peripheral Status Registers (ME_PS*n*) Field Descriptions . . . 183
Table 56. Run Peripheral Configuration Registers (ME_RUN_PC0…7) Field Descriptions. . . . 184
Table 57. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) Field Descriptions. . . 185
Table 58. Peripheral Control Registers (ME_PCTLn) Field Descriptions . . . 185
Table 59. MC_ME Resource Control Overview . . . 191
Table 60. MC_ME System Clock Selection Overview . . . 194
Table 61. MC_PCU memory map. . . . 203
Table 62. MC_PCU registers . . . 203
Table 63. Power Domain Configuration Register Field Descriptions . . . 205
Table 64. Power Domain Status Register (PCU_PSTAT) Field Descriptions. . . . 206
Table 65. MC_RGM Register Description . . . 210
Table 66. MC_RGM Memory Map . . . 211
Table 67. Functional Event Status Register (RGM_FES) Field Descriptions . . . 213
Table 68. Destructive Event Status Register (RGM_DES) Field Descriptions . . . 215
Table 69. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions . . . 216
Table 70. Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions . . . 218
Table 71. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions . . . 218
Table 72. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions . . . 220
Table 73. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions. . . . 221
Table 74. MC_RGM Reset Implications . . . 222
Table 75. MC_RGM Alternate Event Selection . . . 226
Table 76. Interrupt sources available . . . 229
Table 77. INTC memory map . . . 232
Table 78. INTC_MCR field descriptions . . . 233
Table 79. INTC_CPR field descriptions . . . 234
Table 80. INTC_IACKR field descriptions. . . . 235
Table 81. INTC_SSCIR[0:7] field descriptions . . . 238
Table 82. INTC_PSR0_3–INTC_PSR260 field descriptions. . . . 239
Table 83. INTC Priority Select Register address offsets. . . . 239
Table 84. Interrupt vector table. . . . 241
Table 85. Order of ISR execution example. . . . 257
Table 86. SSCM memory map . . . 263
Table 87. STATUS allowed register accesses . . . 264
Table 88. STATUS field descriptions . . . 264
Table 89. MEMCONFIG field descriptions . . . 265
Table 90. MEMCONFIG allowed register accesses . . . 266
Table 91. ERROR field descriptions . . . 266
Table 92. ERROR allowed register accesses. . . . 267
Table 93. DEBUGPORT field descriptions . . . 267
Table 94. Debug Status Port modes. . . . 268
Table 95. DEBUGPORT allowed register accesses . . . 268
Table 96. PWCMPH/L field descriptions. . . . 269
Table 97. PWCMPH/L allowed register accesses . . . 269
Table 98. DPMBOOT Field Descriptions . . . 270
Table 99. DPMKEY Field Descriptions . . . 271
Table 100. UOPS Field Descriptions . . . 271

Table 101. PSA Field Descriptions . . . 272
Table 102. SIUL signal properties . . . 275
Table 103. SIUL memory map . . . 276
Table 104. MIDR1 field descriptions . . . 278
Table 105. MIDR2 field descriptions . . . 279
Table 106. ISR field descriptions . . . 280
Table 107. IRER field descriptions . . . 280
Table 108. IREER field descriptions . . . 281
Table 109. IFEER field descriptions . . . 281
Table 110. IFER field descriptions . . . 282
Table 111. PCR[0:107] field descriptions . . . 283
Table 112. PCR[*n*] reset value exceptions . . . 284
Table 113. PCR bit implementation by pad type . . . 284
Table 114. PSMI[0_3:36_39] field descriptions . . . 285
Table 115. Pad selection . . . 285
Table 116. GPDO[0_3:104_107] field descriptions . . . 288
Table 117. GPDI[0_3:104_107] field descriptions . . . 289
Table 118. PGPDO0_3 field descriptions . . . 289
Table 119. PGPDI[0:3] field descriptions . . . 290
Table 120. MPGPDO[0:6] field descriptions . . . 291
Table 121. IFMC[0:31] field descriptions . . . 291
Table 122. IFCPR field descriptions . . . 292
Table 123. PBRIDGE registers . . . 306
Table 124. PBRIDGE memory map . . . 307
Table 125. MPROT*n* field reset values . . . 308
Table 126. MPROT*n* field structure descriptions . . . 308
Table 127. PACR*n* field structure descriptions . . . 309
Table 128. On-platform peripherals and PACR numbers . . . 309
Table 129. Off-platform peripherals and OPACR numbers . . . 310
Table 130. Logical master IDs . . . 313
Table 131. XBAR master port allocation . . . 314
Table 132. XBAR slave port allocation . . . 314
Table 133. XBAR register configuration summary . . . 316
Table 134. XBAR register summary . . . 317
Table 135. Register terms . . . 318
Table 136. Master Priority Register descriptions . . . 319
Table 137. Slave General Purpose Control Register Descriptions . . . 322
Table 138. Master General Purpose Control Register Descriptions . . . 324
Table 139. ECSM registers . . . 340
Table 140. PCT field descriptions . . . 341
Table 141. REV field descriptions . . . 341
Table 142. PLAMC field descriptions . . . 342
Table 143. ASC field descriptions . . . 342
Table 144. IMC field descriptions . . . 343
Table 145. MRSR field descriptions . . . 344
Table 146. MIR field descriptions . . . 344
Table 147. MUDCR field descriptions . . . 345
Table 148. ECR field descriptions . . . 346
Table 149. ESR field descriptions . . . 348
Table 150. EEGR field descriptions . . . 350
Table 151. FEAR field descriptions . . . 352
Table 152. FEMR field descriptions . . . 352

Table 153. FEAT field descriptions . . . 353
Table 154. FEDR field descriptions . . . 354
Table 155. REAR field descriptions . . . 355
Table 156. RESR field descriptions . . . 355
Table 157. RAM syndrome mapping for single-bit correctable errors . . . 355
Table 158. REMR field descriptions . . . 357
Table 159. REAT field descriptions . . . 358
Table 160. REDR field descriptions . . . 359
Table 161. SRAM operating modes . . . 360
Table 162. SRAM memory map . . . 360
Table 163. Number of wait states required for SRAM operations . . . 361
Table 164. Flash-related regions in the system memory map . . . 367
Table 165. Platform Flash controller 32-bit memory map . . . 368
Table 166. Platform Flash controller stall-while-write interrupts . . . 376
Table 167. Additional wait state encoding . . . 376
Table 168. Extended additional wait state encoding . . . 377
Table 169. Code Flash memory storage address map . . . 387
Table 170. 64 KB data Flash module sectorization . . . 388
Table 171. TestFlash structure . . . 388
Table 172. Shadow sector structure . . . 390
Table 173. Flash registers . . . 393
Table 174. Flash 1056 KB multi-module register map . . . 393
Table 175. Flash 64 KB bank1 register map . . . 395
Table 176. MCR field descriptions . . . 396
Table 177. MCR bits set/clear priority levels . . . 400
Table 178. LML and NVLML field descriptions . . . 402
Table 179. HBL and NVHBL field descriptions . . . 404
Table 180. SLL and NVSLL field descriptions . . . 406
Table 181. LMS field descriptions . . . 408
Table 182. HBS field descriptions . . . 409
Table 183. ADR field descriptions . . . 410
Table 184. ADR content: priority list . . . 411
Table 185. PFCR0 field descriptions . . . 412
Table 186. PFCR1 field descriptions . . . 416
Table 187. PFAPR field descriptions . . . 419
Table 188. BIU2 and NVBIU2 field descriptions . . . 420
Table 189. BIU3 and NVBIU3 field descriptions . . . 421
Table 190. UT0 field descriptions . . . 421
Table 191. UT1 field descriptions . . . 423
Table 192. UT2 field descriptions . . . 424
Table 193. UMSIR0 field descriptions . . . 425
Table 194. UMISR1 field descriptions . . . 425
Table 195. UMISR2 field descriptions . . . 426
Table 196. UMISR3 field descriptions . . . 427
Table 197. UMISR4 field descriptions . . . 427
Table 198. NVPWD0 field descriptions . . . 428
Table 199. NVPWD1 field descriptions . . . 429
Table 200. NVSCI0 field descriptions . . . 429
Table 201. NVSCI1 field descriptions . . . 430
Table 202. NVUSRO field descriptions . . . 431
Table 203. Flash modify operations . . . 432
Table 204. Bits manipulation: double words with the same ECC value . . . 440

Table 205. Bits manipulation: censorship management . . . 442
Table 206. MPU port allocation . . . 445
Table 207. MPU memory map . . . 446
Table 208. MPU_CESR field descriptions . . . 448
Table 209. MPU_EARn field descriptions . . . 449
Table 210. MPU_EDRn field descriptions . . . 450
Table 211. MPU_RGDn.Word0 field descriptions . . . 451
Table 212. MPU_RGDn.Word1 field descriptions . . . 451
Table 213. MPU_RGDn.Word2 field descriptions . . . 452
Table 214. MPU_RGDn.Word3 field descriptions . . . 454
Table 215. MPU_RGDAACn field descriptions . . . 455
Table 216. Protection violation definition . . . 458
Table 217. Overlapping region descriptor example . . . 461
Table 218. eDMA memory map . . . 465
Table 219. EDMA_CR field descriptions . . . 468
Table 220. EDMA_ESR field descriptions . . . 469
Table 221. EDMA_ERQRL field descriptions . . . 471
Table 222. EDMA_EEIRL field descriptions . . . 472
Table 223. EDMA_SERQR field descriptions . . . 472
Table 224. EDMA_CERQR field descriptions . . . 473
Table 225. EDMA_SEEIR field descriptions . . . 474
Table 226. EDMA_CEEIR field descriptions . . . 474
Table 227. EDMA_CIRQR field descriptions . . . 475
Table 228. EDMA_CER field descriptions . . . 475
Table 229. EDMA_SSBR field descriptions . . . 476
Table 230. EDMA_CDSBR field descriptions . . . 477
Table 231. EDMA_IRQRL field descriptions . . . 477
Table 232. EDMA_ERL field descriptions . . . 478
Table 233. EDMA_HRSL field descriptions . . . 479
Table 234. EDMA_CPR*n* field descriptions . . . 480
Table 235. TCDn 32-bit memory structure . . . 480
Table 236. TCD*n* field descriptions . . . 482
Table 237. eDMA peak transfer rates (MB/Sec) . . . 492
Table 238. eDMA peak request Rate (MReq/sec) . . . 493
Table 239. TCD primary control and status fields . . . 495
Table 240. DMA request summary for eDMA . . . 497
Table 241. Modulo feature example . . . 501
Table 242. Channel linking parameters . . . 504
Table 243. DMA_MUX memory map . . . 506
Table 244. CHCONFIG#*x* field descriptions . . . 508
Table 245. Channel and trigger enabling . . . 508
Table 246. DMA channel mapping . . . 509
Table 247. External signal properties . . . 522
Table 248. FlexRay memory map . . . 524
Table 249. Register access conventions . . . 527
Table 250. Additional register reset conditions . . . 527
Table 251. Register write access restrictions . . . 528
Table 252. MVR field descriptions . . . 529
Table 253. MCR field descriptions . . . 529
Table 254. FlexRay channel selection . . . 530
Table 255. FlexRay channel bit rate selection . . . 531
Table 256. SYMBADHR and SYMBADLR field descriptions . . . 531

Table 257. STBSCR field descriptions . . . 532
Table 258. Strobe signal mapping . . . 533
Table 259. MBDSR field descriptions . . . 535
Table 260. MBSSUTR field descriptions . . . 536
Table 261. POCR field descriptions . . . 537
Table 262. GIFER field descriptions . . . 538
Table 263. PIFR0 field description . . . 541
Table 264. PIFR1 field descriptions . . . 543
Table 265. PIER0 field descriptions . . . 544
Table 266. PIER1 field descriptions . . . 546
Table 267. CHIERFR field descriptions . . . 547
Table 268. MBIVEC field descriptions . . . 549
Table 269. CASERCR field descriptions . . . 549
Table 270. CBSERCR field descriptions . . . 550
Table 271. PSR0 field descriptions . . . 551
Table 272. PSR1 field descriptions . . . 552
Table 273. PSR2 field descriptions . . . 553
Table 274. PSR3 field descriptions . . . 555
Table 275. MTCTR field descriptions . . . 556
Table 276. CYCTR field descriptions . . . 557
Table 277. SLTCTAR field descriptions . . . 557
Table 278. SLTCTBR field descriptions . . . 558
Table 279. RTCORVR field descriptions . . . 558
Table 280. OFCORVR field descriptions . . . 559
Table 281. CIFRR field descriptions . . . 560
Table 282. SYMATOR field descriptions . . . 560
Table 283. SFCNTR field descriptions . . . 561
Table 284. SFTOR field description . . . 562
Table 285. SFTCCSR field descriptions . . . 562
Table 286. SFIDRFR field descriptions . . . 563
Table 287. SFIDAFVR field descriptions . . . 564
Table 288. SFIDAFMR field descriptions . . . 564
Table 289. NMVR[0:5] field descriptions . . . 565
Table 290. Mapping of NMVRn to received payload bytes NMVn . . . 565
Table 291. NMVLR field descriptions . . . 566
Table 292. TICCR field descriptions . . . 566
Table 293. TI1CYSR field descriptions . . . 567
Table 294. TI1MTOR field descriptions . . . 568
Table 295. TI2CR0 field descriptions . . . 568
Table 296. TI2CR1 field descriptions . . . 569
Table 297. SSSR field descriptions . . . 570
Table 298. Mapping between SSSRn and SSRn . . . 570
Table 299. SSCCR field descriptions . . . 571
Table 300. Mapping between internal SSCCRn and SSCRn . . . 572
Table 301. SSR0–SSR7 field descriptions . . . 572
Table 302. SSCR0–SSCR3 field descriptions . . . 574
Table 303. MTSACFR field descriptions . . . 575
Table 304. MTSBCFR field descriptions . . . 575
Table 305. RSBIR field descriptions . . . 576
Table 306. SEL controlled receiver FIFO registers . . . 576
Table 307. RFSR field descriptions . . . 577
Table 308. RFSIR field descriptions . . . 577

Table 309. RFDSR field descriptions . . . 577
Table 310. RFARIR field descriptions . . . 578
Table 311. RFBRIR field descriptions . . . 578
Table 312. RFMIDAFVR field descriptions . . . 579
Table 313. RFMIAFMR field descriptions . . . 579
Table 314. RFFIDRFVR field descriptions . . . 580
Table 315. RFFIDRFMR field descriptions . . . 580
Table 316. RFRFCFR field descriptions . . . 581
Table 317. RFRFCTR field descriptions . . . 581
Table 318. LDTXSLAR field descriptions . . . 582
Table 319. LDTXSLBR field descriptions . . . 583
Table 320. Protocol configuration register fields . . . 583
Table 321. Wakeup channel selection . . . 585
Table 322. MBCCSR*n* field descriptions . . . 592
Table 323. MBCCFR*n* field descriptions . . . 594
Table 324. Channel assignment description . . . 594
Table 325. MBFIDRn field descriptions . . . 595
Table 326. MBIDXR*n* field descriptions . . . 595
Table 327. Frame header write access constraints (Transmit message buffer) . . . 605
Table 328. Frame header field descriptions (Receive message buffer and receive FFO) . . . 606
Table 329. Frame header field descriptions (Transmit message buffer) . . . 607
Table 330. Receive message buffer slot status content . . . 608
Table 331. Receive message buffer slot status field descriptions . . . 608
Table 332. Transmit message buffer slot status content . . . 610
Table 333. Transmit message buffer slot status structure field descriptions . . . 611
Table 334. Message buffer data field minimum length . . . 612
Table 335. Frame data write access constraints . . . 613
Table 336. Frame data field descriptions . . . 613
Table 337. Individual message buffer types . . . 615
Table 338. Single transmit message buffer access regions description . . . 616
Table 339. Single transmit message buffer state description . . . 617
Table 340. Single transmit message buffer application transitions . . . 619
Table 341. Single transmit message buffer module transitions . . . 619
Table 342. Single transmit message buffer transition priorities . . . 620
Table 343. Receive message buffer access region description . . . 624
Table 344. Receive message buffer states and access . . . 625
Table 345. Receive message buffer application transitions . . . 626
Table 346. Receive message buffer module transitions . . . 627
Table 347. Receive message buffer transition priorities . . . 627
Table 348. Receive message buffer update . . . 628
Table 349. Double transmit message buffer access regions description . . . 631
Table 350. Double transmit message buffer state description (commit side) . . . 633
Table 351. Double transmit message buffer state description (transmit side) . . . 634
Table 352. Double Transmit Message Buffer Host Transitions . . . 636
Table 353. Double Transmit Message Buffer Module Transitions . . . 636
Table 354. Double transmit message buffer transition priorities . . . 637
Table 355. Message buffer search priority (static segment) . . . 640
Table 356. Message buffer search priority (dynamic segment) . . . 640
Table 357. Sync frame table generation modes . . . 652
Table 358. Key slot frame type . . . 654
Table 359. Slot status content . . . 658
Table 360. FlexRay Channel Bit Rate Control . . . 665

Table 361. Minimum f_{chi} [MHz] examples (32 message buffers) . . . 667
Table 362. Protocol control command priorities . . . 668
Table 363. Transmit buffer configuration . . . 669
Table 364. Receive buffer configuration . . . 669
Table 365. Signal properties . . . 675
Table 366. DSPI memory map . . . 677
Table 367. DSPIx_MCR field descriptions . . . 679
Table 368. DSPIx_TCR field descriptions . . . 682
Table 369. DSPIx_CTARn field descriptions . . . 683
Table 370. DSPI SCK duty cycle . . . 686
Table 371. DSPI transfer frame size . . . 686
Table 372. DSPI PCS to SCK delay scaler . . . 687
Table 373. DSPI after SCK delay scaler . . . 687
Table 374. DSPI delay after transfer scaler . . . 687
Table 375. DSPI baud rate scaler . . . 688
Table 376. DSPIx_SR field descriptions . . . 688
Table 377. DSPI*x*_RSER field descriptions . . . 690
Table 378. DSPIx_PUSHR field descriptions . . . 692
Table 379. DSPIx_POPR field descriptions . . . 694
Table 380. DSPIx_TXFRn field descriptions . . . 695
Table 381. DSPIx_RXFRn field description . . . 695
Table 382. State transitions for start and stop of DSPI transfers . . . 698
Table 383. Baud rate computation example . . . 702
Table 384. CS to SCK delay computation example . . . 702
Table 385. After SCK delay computation example . . . 702
Table 386. Delay after transfer computation example . . . 703
Table 387. Peripheral Chip Select strobe assert computation example . . . 704
Table 388. Peripheral Chip Select strobe negate computation example . . . 704
Table 389. Delayed master sample point . . . 707
Table 390. Interrupt and DMA request conditions . . . 713
Table 391. Baud rate values . . . 716
Table 392. Delay values . . . 717
Table 393. Error calculation for programmed baud rates . . . 723
Table 394. LINFlex memory map . . . 726
Table 395. LINCR1 field descriptions . . . 728
Table 396. Checksum bits configuration . . . 729
Table 397. LIN master break length selection . . . 729
Table 398. Operating mode selection . . . 730
Table 399. LINIER field descriptions . . . 731
Table 400. LINSR field descriptions . . . 733
Table 401. LINESR field descriptions . . . 735
Table 402. UARTCR field descriptions . . . 737
Table 403. UARTSR field descriptions . . . 738
Table 404. LINTCSR field descriptions . . . 740
Table 405. LINOCR field descriptions . . . 741
Table 406. LINTOCR field descriptions . . . 742
Table 407. LINFBRR field descriptions . . . 742
Table 408. LINIBRR field descriptions . . . 743
Table 409. Integer baud rate selection . . . 743
Table 410. LINCFR field descriptions . . . 744
Table 411. LINCR2 field descriptions . . . 745
Table 412. BIDR field descriptions . . . 746

Table 413. BDRL field descriptions . . . 747
Table 414. BDRM field descriptions . . . 748
Table 415. IFER field descriptions . . . 748
Table 416. IFMI field descriptions . . . 749
Table 417. IFMR field descriptions . . . 750
Table 418. IFMR[IFM] configuration . . . 750
Table 419. IFCR2*n* field descriptions . . . 751
Table 420. IFCR2*n* + 1 field descriptions . . . 752
Table 421. Message buffer . . . 754
Table 422. Filter to interrupt vector correlation . . . 760
Table 423. LINFlex interrupt control . . . 764
Table 424. FlexCAN signals . . . 769
Table 425. FlexCAN module memory map . . . 770
Table 426. FlexCAN register reset status . . . 770
Table 427. Message Buffer MB0 memory mapping . . . 771
Table 428. Message Buffer structure field description . . . 772
Table 429. Message buffer code for Rx buffers . . . 773
Table 430. Message Buffer code for Tx buffers . . . 774
Table 431. MB0–MB31 addresses . . . 774
Table 432. ID Table 0 - 7 . . . 776
Table 433. Rx FIFO Structure field description . . . 777
Table 434. MCR field descriptions . . . 778
Table 435. IDAM coding . . . 781
Table 436. CTRL field descriptions . . . 782
Table 437. TIMER field descriptions . . . 785
Table 438. RXGMASK field description . . . 786
Table 439. RX14MASK field description . . . 786
Table 440. RX15MASK field description . . . 787
Table 441. Error and Status Register (ESR) field description . . . 789
Table 442. Fault confinement state . . . 791
Table 443. IMASK1 field descriptions . . . 792
Table 444. IFLAG1 field descriptions . . . 792
Table 445. RXIMR0–RXIMR31 field descriptions . . . 794
Table 446. RXIMR0–RXIMR31 addresses . . . 794
Table 447. Time segment syntax . . . 805
Table 448. CAN standard compliant bit time segment settings . . . 805
Table 449. Minimum ratio between peripheral clock frequency and CAN bit rate . . . 806
Table 450. Configurations for starting normal conversion . . . 812
Table 451. ADC sampling and conversion timing at 5 V / 3.3 V for ADC0 . . . 817
Table 452. Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC0 . . . 818
Table 453. Presampling voltage selection based on PREVALx fields . . . 819
Table 454. Values of WDGxH and WDGxL fields . . . 820
Table 455. Example for Analog watchdog operation . . . 821
Table 456. ADC digital registers . . . 823
Table 457. MCR field descriptions . . . 825
Table 458. MSR field descriptions . . . 827
Table 459. ISR field descriptions . . . 828
Table 460. IMR field descriptions . . . 829
Table 461. WTISR field descriptions . . . 830
Table 462. WTIMR field descriptions . . . 831
Table 463. DMAE field descriptions . . . 832
Table 464. DMARx field descriptions . . . 833

Table 465. TRC*x* field descriptions . . . 834
Table 466. THRHLR*x* field descriptions . . . 835
Table 467. PSCR field descriptions . . . 836
Table 468. PSR field descriptions . . . 836
Table 469. CTR field descriptions . . . 837
Table 470. NCMR field descriptions . . . 838
Table 471. JCMR field descriptions . . . 838
Table 472. PDEDR field descriptions . . . 839
Table 473. CDR field descriptions . . . 840
Table 474. ADC commands translation . . . 849
Table 475. CTU interrupts . . . 854
Table 476. CTU memory map . . . 855
Table 477. TGS registers . . . 857
Table 478. SU registers . . . 858
Table 479. CTU registers . . . 858
Table 480. FIFO registers . . . 858
Table 481. TGSISR field descriptions . . . 859
Table 482. TGSCR field descriptions . . . 862
Table 483. TxCR field descriptions . . . 862
Table 484. TGSCCR field format . . . 862
Table 485. TGSCRR field descriptions . . . 863
Table 486. CLCR1 field descriptions . . . 863
Table 487. CLCR2 field descriptions . . . 864
Table 488. THCR1 field descriptions . . . 865
Table 489. THCR2 field descriptions . . . 866
Table 490. CLR*x* (CMS = 0) field descriptions . . . 868
Table 491. CLRx (CMS = 1) field descriptions . . . 869
Table 492. FDCR field descriptions . . . 870
Table 493. FCR field descriptions . . . 870
Table 494. FTH field descriptions . . . 872
Table 495. FST field descriptions . . . 873
Table 496. FR*x* field descriptions . . . 874
Table 497. FL*x* field descriptions . . . 875
Table 498. CTUEFR field descriptions . . . 875
Table 499. CTUIFR field descriptions . . . 876
Table 500. CTUIR field descriptions . . . 877
Table 501. COTR field descriptions . . . 878
Table 502. CTUCR field descriptions . . . 879
Table 503. CTUDF field descriptions . . . 880
Table 504. CTUPCR field descriptions . . . 880
Table 505. SEMA4 memory map . . . 883
Table 506. SEMA4_GATE*n* field descriptions . . . 885
Table 507. SEMA4_CP{0,1}NTF field descriptions . . . 886
Table 508. SEMA4_CP{0,1}NTF field descriptions . . . 886
Table 509. SEMA4_RSTGT field descriptions . . . 887
Table 510. SEMA4_RSTNTF field descriptions . . . 889
Table 511. eTimer memory map . . . 898
Table 512. COMP1 field descriptions . . . 902
Table 513. COMP2 field descriptions . . . 902
Table 514. CAPT1 field descriptions . . . 903
Table 515. CAPT2 field descriptions . . . 903
Table 516. LOAD field descriptions . . . 904

Table 517. HOLD field descriptions . . . 904
Table 518. CNTR field descriptions . . . 905
Table 519. CTRL1 field descriptions . . . 905
Table 520. Count source values . . . 906
Table 521. CTRL2 field descriptions . . . 907
Table 522. CTRL3 field descriptions . . . 910
Table 523. STS field descriptions . . . 911
Table 524. INTDMA field descriptions . . . 912
Table 525. CMPLD1 field descriptions . . . 913
Table 526. CMPLD2 field descriptions . . . 914
Table 527. CCCTRL field descriptions . . . 914
Table 528. FILT field descriptions . . . 916
Table 529. WDTOL, WDTOH field descriptions . . . 917
Table 530. ENBL field descriptions . . . 918
Table 531. DREQ*n* field descriptions . . . 919
Table 532. Interrupt summary . . . 926
Table 533. DMA summary . . . 927
Table 534. Register protection memory map . . . 930
Table 535. SLBR*n* field descriptions . . . 931
Table 536. Soft Lock Bits vs. Protected Address . . . 932
Table 537. GCR field descriptions . . . 932
Table 538. SWT memory map . . . 938
Table 539. SWT_CR field descriptions . . . 939
Table 540. SWT_IR field descriptions . . . 940
Table 541. SWT_TO field descriptions . . . 941
Table 542. SWT_WN field descriptions . . . 941
Table 543. SWT_SR field descriptions . . . 942
Table 544. SWT_CO field descriptions . . . 943
Table 545. SWT_SR field descriptions . . . 943
Table 546. Acronyms . . . 946
Table 547. FCCU memory map . . . 949
Table 548. FCCU_CTRL field descriptions . . . 952
Table 549. FCCU_CTRLK field descriptions . . . 954
Table 550. FCCU_CFG field descriptions . . . 955
Table 551. FCCU_CF_CFG0..3 field descriptions . . . 957
Table 552. FCCU_NCF_CFG0..3 field descriptions . . . 958
Table 553. FCCU_CFS_CFG0..7 field descriptions . . . 960
Table 554. FCCU_NCFS_CFG0..7 field descriptions . . . 961
Table 555. FCCU_CFS0..3 field descriptions . . . 963
Table 556. FCCU_CFK field descriptions . . . 963
Table 557. FCCU_NCFS0..3 field descriptions . . . 965
Table 558. FCCU_NCFK field descriptions . . . 966
Table 559. FCCU_NCFE0..3 field descriptions . . . 967
Table 560. FCCU_NCF_TOE0..3 field descriptions . . . 967
Table 561. FCCU_NCF_TO field descriptions . . . 968
Table 562. FCCU_CFG_TO field descriptions . . . 969
Table 563. FCCU_STAT field descriptions . . . 970
Table 564. FCCU_CFF field descriptions . . . 971
Table 565. FCCU_NCFF field descriptions . . . 972
Table 566. FCCU_IRQ_STAT field descriptions . . . 973
Table 567. FCCU_IRQ_EN field descriptions . . . 974
Table 568. Timer state/value . . . 974

Table 569. FCCU_XTMR field descriptions . . . 975
Table 570. FCCU_MCS field descriptions . . . 976
Table 571. Reset sources . . . 979
Table 572. NVM configuration . . . 984
Table 573. Dual-rail encoding . . . 986
Table 574. Time switching encoding . . . 987
Table 575. Bi-stable encoding . . . 989
Table 576. FCCU mapping of critical faults . . . 989
Table 577. FCCU mapping of non-critical faults . . . 990
Table 578. WKPU memory map . . . 991
Table 579. NSR field descriptions . . . 992
Table 580. NCR field descriptions . . . 993
Table 581. PIT memory map . . . 997
Table 582. PITMCR field descriptions . . . 998
Table 583. LDVAL*n* field descriptions . . . 999
Table 584. CVAL*n* field descriptions . . . 1000
Table 585. TCTRL*n* field descriptions . . . 1001
Table 586. TFLG*n* field descriptions . . . 1002
Table 587. STM memory map . . . 1006
Table 588. STM_CR field descriptions . . . 1007
Table 589. STM_CNT field descriptions . . . 1008
Table 590. STM_CCR*n* field descriptions . . . 1008
Table 591. STM_CIR*n* field descriptions . . . 1009
Table 592. STM_CMP*n* field descriptions . . . 1009
Table 593. CRC memory map . . . 1014
Table 594. CRC_CFG field descriptions . . . 1016
Table 595. CRC_INP field descriptions . . . 1017
Table 596. CRC_CSTAT field descriptions . . . 1017
Table 597. CRC_OUTP field descriptions . . . 1018
Table 598. CRC_OUTP_CHK field descriptions . . . 1019
Table 599. BAM memory organization . . . 1023
Table 600. Hardware configuration to select boot mode . . . 1025
Table 601. SPC56xP60/54 boot pins . . . 1025
Table 602. RCHW field descriptions . . . 1026
Table 603. Flash boot sector . . . 1027
Table 604. Fields of SSCM STATUS register used by BAM . . . 1030
Table 605. Serial boot mode without autobaud—baud rates . . . 1030
Table 606. UART boot mode download protocol (autobaud disabled) . . . 1035
Table 607. FlexCAN boot mode download protocol (autobaud disabled) . . . 1036
Table 608. System clock frequency related to external clock frequency . . . 1037
Table 609. Maximum and minimum recommended baud rates . . . 1041
Table 610. Prescaler/divider and time base values . . . 1045
Table 611. FlexCAN standard compliant bit timing segment settings . . . 1046
Table 612. Lookup table for FlexCAN bit timings . . . 1046
Table 613. PRESDIV + 1 = 1 . . . 1046
Table 614. PRESDIV + 1 > 1 (YY = PRESDIV) . . . 1047
Table 615. Examples of legal and illegal passwords . . . 1049
Table 616. Censorship configuration and truth table . . . 1050
Table 617. VREG_CTL field descriptions . . . 1055
Table 618. VREG_STATUS field descriptions . . . 1056
Table 619. JTAG signal properties . . . 1060
Table 620. Device identification register field descriptions . . . 1062

Table 621. JTAG instructions . . . 1065
Table 622. e200z0 OnCE register addressing . . . 1069
Table 623. Device parameter values . . . 1071
Table 624. NPC Signal Properties . . . 1074
Table 625. NPC Registers . . . 1076
Table 626. DID Field Descriptions . . . 1077
Table 627. PCR field descriptions . . . 1078
Table 628. MCKO_DIV Values . . . 1080
Table 629. NPC Reset Configuration Options . . . 1080
Table 630. NPC Output Messages . . . 1082
Table 631. Implemented Instructions . . . 1083
Table 632. Loading NEXUS-ENABLE instruction . . . 1086
Table 633. Write to a 32-Bit Nexus Client Register . . . 1087
Table 634. Registers under protection . . . 1090
Table 635. Revision history . . . 1103

List of figures

Figure 1. Airbag application . . . 54
Figure 2. SPC56xP60/54 block diagram . . . 57
Figure 3. LQFP176 pinout (top view) . . . 79
Figure 4. LQFP144 pinout (top view) . . . 80
Figure 5. LQFP100 pinout (top view) . . . 81
Figure 6. CTU / ADCs / eTimers / DSPI / FlexRay connections . . . 100
Figure 7. SPC56xP60/54 system clock generation . . . 103
Figure 8. SPC56xP60/54 system clock distribution Part A . . . 104
Figure 9. SPC56xP60/54 system clock distribution Part B . . . 105
Figure 10. Output Clock Division 256 Select Register (CLK_DIV_256) . . . 107
Figure 11. RC Control register (RC_CTL) . . . 111
Figure 12. Crystal Oscillator Control register (OSC_CTL) . . . 113
Figure 13. FMPLL block diagram . . . 114
Figure 14. Control Register (CR) . . . 115
Figure 15. Modulation Register (MR) . . . 117
Figure 16. Progressive clock switching scheme . . . 119
Figure 17. Frequency modulation depth spreads . . . 121
Figure 18. SPC56xP60/54 with two CMUs . . . 123
Figure 19. Control Status Register (CMU_0_CSR) . . . 126
Figure 20. Frequency Display Register (CMU_0_FDR) . . . 127
Figure 21. High Frequency Reference register FMPLL_0 (CMU_0_HFREFR_0) . . . 127
Figure 22. Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_0) . . . 128
Figure 23. Interrupt Status Register (CMU_0_ISR) . . . 128
Figure 24. Measurement Duration Register (CMU_0_MDR) . . . 129
Figure 25. Control Status Register (CMU_1_CSR) . . . 130
Figure 26. High Frequency Reference Register SYS_CLK (CMU_1_HFREFR_0) . . . 130
Figure 27. Low Frequency Reference Register SYS_CLK (CMU_1_LFREFR_0) . . . 131
Figure 28. Interrupt Status register (CMU_1_ISR) . . . 131
Figure 29. MC_CGMBlock Diagram . . . 134
Figure 30. Output Clock Enable Register (CGM_OC_EN) . . . 139
Figure 31. Output Clock Division Select Register (CGM_OCDS_SC) . . . 140
Figure 32. System Clock Select Status Register (CGM_SC_SS) . . . 141
Figure 33. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) . . . 142
Figure 34. Auxiliary Clock 0 Divider 0 Configuration Register (CGM_AC0_DC0) . . . 142
Figure 35. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) . . . 143
Figure 36. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0) . . . 144
Figure 37. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) . . . 145
Figure 38. Auxiliary Clock 2 Divider 0 Configuration Register (CGM_AC2_DC0) . . . 145
Figure 39. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) . . . 146
Figure 40. Auxiliary Clock 3 Divider 0 Configuration Register (CGM_AC3_DC0) . . . 147
Figure 41. MC_CGM System Clock Generation Overview . . . 148
Figure 42. MC_CGM Auxiliary Clock 0 Generation Overview . . . 149
Figure 43. MC_CGM Auxiliary Clock 1 Generation Overview . . . 149
Figure 44. MC_CGM Auxiliary Clock 2 Generation Overview . . . 150
Figure 45. MC_CGM Auxiliary Clock 3 Generation Overview . . . 150
Figure 46. MC_CGM Output Clock Multiplexer and PAD[22] Generation . . . 152
Figure 47. MC_ME Block Diagram . . . 154
Figure 48. Global Status Register (ME_GS) . . . 165

Figure 49. Mode Control Register (ME_MCTL) . . . 167
Figure 50. Mode Enable Register (ME_ME) . . . 168
Figure 51. Interrupt Status Register (ME_IS) . . . 170
Figure 52. Interrupt Mask Register (ME_IM) . . . 171
Figure 53. Invalid Mode Transition Status Register (ME_IMTS) . . . 172
Figure 54. Debug Mode Transition Status Register (ME_DMTS) . . . 173
Figure 55. RESET Mode Configuration Register (ME_RESET_MC) . . . 176
Figure 56. TEST Mode Configuration Register (ME_TEST_MC) . . . 177
Figure 57. SAFE Mode Configuration Register (ME_SAFE_MC) . . . 177
Figure 58. DRUN Mode Configuration Register (ME_DRUN_MC) . . . 178
Figure 59. RUN0…3 Mode Configuration Registers (ME_RUN0...3_MC) . . . 178
Figure 60. HALT0 Mode Configuration Register (ME_HALT0_MC) . . . 179
Figure 61. STOP0 Mode Configuration Register (ME_STOP0_MC) . . . 179
Figure 62. Peripheral Status Register 0 (ME_PS0) . . . 181
Figure 63. Peripheral Status Register 1 (ME_PS1) . . . 182
Figure 64. Peripheral Status Register 2 (ME_PS2) . . . 182
Figure 65. Peripheral Status Register 3 (ME_PS3) . . . 183
Figure 66. Run Peripheral Configuration Registers (ME_RUN_PC0...7) . . . 183
Figure 67. Low-Power Peripheral Configuration Registers (ME_LP_PC0...7) . . . 184
Figure 68. Peripheral Control Registers (ME_PCTLn) . . . 185
Figure 69. MC_ME Mode Diagram . . . 187
Figure 70. MC_ME Transition Diagram . . . 197
Figure 71. MC_ME Application Example Flow Diagram . . . 201
Figure 72. MC_PCU Block Diagram . . . 202
Figure 73. Power Domain #0 Configuration Register (PCU_PCONF0) . . . 205
Figure 74. Power Domain Status Register (PCU_PSTAT) . . . 206
Figure 75. MC_RGM Block Diagram . . . 208
Figure 76. Functional Event Status Register (RGM_FES) . . . 213
Figure 77. Destructive Event Status Register (RGM_DES) . . . 214
Figure 78. Functional Event Reset Disable Register (RGM_FERD) . . . 216
Figure 79. Destructive Event Reset Disable Register (RGM_DERD) . . . 217
Figure 80. Functional Event Alternate Request Register (RGM_FEAR) . . . 218
Figure 81. Functional Event Short Sequence Register (RGM_FESS) . . . 219
Figure 82. Functional Bidirectional Reset Enable Register (RGM_FBRE) . . . 221
Figure 83. MC_RGM State Machine . . . 223
Figure 84. INTC block diagram . . . 230
Figure 85. INTC Module Configuration Register (INTC_MCR) . . . 233
Figure 86. INTC Current Priority Register (INTC_CPR) . . . 233
Figure 87. INTC Interrupt Acknowledge Register (INTC_IACKR) . . . 235
Figure 88. INTC Interrupt Acknowledge Register 1 (INTC_IACKR1) . . . 236
Figure 89. INTC End-of-Interrupt Register (INTC_EOIR) . . . 236
Figure 90. INTC End-of-Interrupt Register 1 (INTC_EOIR1) . . . 237
Figure 91. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3]) . . . 237
Figure 92. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7]) . . . 237
Figure 93. INTC Priority Select Register 0–3 (INTC_PSR[0:3]) . . . 238
Figure 94. INTC Priority Select Register 260 (INTC_PSR[260]) . . . 239
Figure 95. Software vector mode handshaking timing diagram . . . 253
Figure 96. Hardware vector mode handshaking timing diagram . . . 254
Figure 97. SSCM block diagram . . . 262
Figure 98. Key to register fields . . . 264
Figure 99. Status (STATUS) register . . . 264
Figure 100. System memory configuration (MEMCONFIG) register . . . 265

Figure 101. Error Configuration (ERROR) register . . . 266
Figure 102. Debug Status Port (DEBUGPORT) register . . . 267
Figure 103. Password Comparison Register High Word (PWCMPH) register . . . 268
Figure 104. Password Comparison Register Low Word (PWCMPL) register . . . 269
Figure 105. DPM Boot (DPMBOOT) Register . . . 270
Figure 106. Boot Key Register (DPMKEY) Register . . . 270
Figure 107. User Option Status (UOPS) Register . . . 271
Figure 108. Processor Start Address (PSA) Register . . . 272
Figure 109. System Integration Unit Lite block diagram . . . 274
Figure 110. Key to register fields . . . 277
Figure 111. MCU ID Register #1 (MIDR1) . . . 278
Figure 112. MCU ID Register #2 (MIDR2) . . . 279
Figure 113. Interrupt Status Flag Register (ISR) . . . 280
Figure 114. Interrupt Request Enable Register (IRER) . . . 280
Figure 115. Interrupt Rising-Edge Event Enable Register (IREER) . . . 281
Figure 116. Interrupt Falling-Edge Event Enable Register (IFEER) . . . 281
Figure 117. Interrupt Filter Enable Register (IFER) . . . 282
Figure 118. Pad Configuration Registers 0–108 (PCR[0:108]) . . . 282
Figure 119. Pad Selection for Multiplexed Inputs registers (PSMI[0_3:36_39]) . . . 285
Figure 120. Port GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107]) . . . 288
Figure 121. GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107]) . . . 288
Figure 122. Parallel GPIO Pad Data Out register 0–3(PGPDO[0:3]) . . . 289
Figure 123. Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3]) . . . 290
Figure 124. Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6]) . . . 290
Figure 125. Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31]) . . . 291
Figure 126. Interrupt Filter Clock Prescaler Register (IFCPR) . . . 292
Figure 127. Data port example arrangement showing configuration for different port width accesses 293
Figure 128. External interrupt pad diagram . . . 294
Figure 129. e200z0 block diagram . . . 298
Figure 130. e200z0h block diagram . . . 299
Figure 131. e200z0 Supervisor mode programmer's model . . . 302
Figure 132. e200z0h Supervisor mode programmer's model . . . 303
Figure 133. e200 User mode program model . . . 304
Figure 134. PBRIDGE interface . . . 305
Figure 135. MPROT*n* field structure . . . 308
Figure 136. PACR*n* field structure . . . 309
Figure 137. Key to Register Fields . . . 317
Figure 138. Master Priority Register n . . . 319
Figure 139. Slave General Purpose Control Register n . . . 322
Figure 140. Master General Purpose Control Register n . . . 324
Figure 141. Low to high priority mastership change . . . 332
Figure 142. High to low priority mastership change . . . 333
Figure 143. Round-robin mastership change . . . 333
Figure 144. Parking on a specific master . . . 335
Figure 145. Parking on last master . . . 336
Figure 146. Processor core type (PCT) register . . . 341
Figure 147. Revision (REV) register . . . 341
Figure 148. Platform XBAR Master Configuration (PLAMC) register . . . 342
Figure 149. Platform XBAR Slave Configuration (PLASC) register . . . 342
Figure 150. IPS Module Configuration (IMC) register . . . 343
Figure 151. Miscellaneous Reset Status Register (MRSR) . . . 343
Figure 152. Miscellaneous Interrupt Register (MIR) . . . 344

Figure 153. Miscellaneous User-Defined Control register (MUDCR) . . . 345
Figure 154. ECC Configuration register (ECR) . . . 346
Figure 155. ECC Status register (ESR) . . . 348
Figure 156. ECC Error Generation register (EEGR) . . . 349
Figure 157. Flash ECC Address register (FEAR) . . . 352
Figure 158. Flash ECC Master Number Register (FEMR) . . . 352
Figure 159. Flash ECC Attributes (FEAT) Register . . . 353
Figure 160. Flash ECC Data register (FEDR) . . . 354
Figure 161. RAM ECC Address register (REAR) . . . 354
Figure 162. RAM ECC Syndrome Register (RESR) . . . 355
Figure 163. RAM ECC Master Number register (REMR) . . . 357
Figure 164. RAM ECC Attributes (REAT) register . . . 358
Figure 165. Platform RAM ECC Data register (PREDR) . . . 359
Figure 166. SPC56xP60/54 Flash memory architecture . . . 364
Figure 167. 1-cycle access, no buffering, no prefetch . . . 378
Figure 168. 3-cycle access, no prefetch, buffering disabled . . . 379
Figure 169. 3-cycle access, no prefetch, buffering enabled . . . 380
Figure 170. 3-cycle access, prefetch and buffering enabled . . . 381
Figure 171. 3-cycle access, stall-and-retry with BK*n*_RWWC = 11*x* . . . *382*
Figure 172. 3-cycle access, terminate-and-retry with BK*n*_RWWC = 10*x* . . . *383*
Figure 173. Data Flash module structure . . . 385
Figure 174. Code Flash module structure . . . 386
Figure 175. Module Configuration Register (MCR) . . . 396
Figure 176. Low/Mid Address Space Block Locking register (LML) . . . 401
Figure 177. Non-Volatile Low/Mid Address Space Block Locking register (NVLML) . . . 401
Figure 178. High Address Space Block Locking register (HBL) . . . 404
Figure 179. Non-Volatile High Address Space Block Locking register (NVHBL) . . . 404
Figure 180. Secondary Low/mid address space block Locking reg (SLL) . . . 405
Figure 181. Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL) . . . 406
Figure 182. Low/Mid Address Space Block Select register (LMS) . . . 408
Figure 183. High Address Space Block Select register (HBS) . . . 409
Figure 184. Address Register (ADR) . . . 410
Figure 185. Platform Flash Configuration Register 0 (PFCR0) . . . 412
Figure 186. Platform Flash Configuration Register 1 (PFCR1) . . . 416
Figure 187. Platform Flash Access Protection Register (PFAPR) . . . 418
Figure 188. Non-Volatile Bus Interface Unit 2 register (NVBIU2) . . . 419
Figure 189. Bus Interface Unit 3 register (BIU3) . . . 420
Figure 190. Non-Volatile Bus Interface Unit 3 register (NVBIU3) . . . 420
Figure 191. User Test 0 register (UT0) . . . 421
Figure 192. User Test 1 register (UT1) . . . 423
Figure 193. User Test 2 register (UT2) . . . 424
Figure 194. User Multiple Input Signature Register 0 (UMISR0) . . . 424
Figure 195. User Multiple Input Signature Register 1 (UMISR1) . . . 425
Figure 196. User Multiple Input Signature Register 2 (UMISR2) . . . 426
Figure 197. User Multiple Input Signature Register 3 (UMISR3) . . . 426
Figure 198. User Multiple Input Signature Register 4 (UMISR4) . . . 427
Figure 199. Non-Volatile private Censorship Password 0 register (NVPWD0) . . . 428
Figure 200. Non-Volatile Private Censorship Password 1 register (NVPWD1) . . . 428
Figure 201. Non-Volatile System Censoring Information 0 register (NVSCI0) . . . 429
Figure 202. Non-Volatile System Censoring Information 1 register (NVSCI1) . . . 430
Figure 203. Non-Volatile User Options register (NVUSRO) . . . 431
Figure 204. MPU block diagram . . . 444

Figure 205. MPU Control/Error Status Register (MPU_CESR) . . . 448
Figure 206. MPU Error Address Register, Slave Port n (MPU_EARn) . . . 449
Figure 207. MPU Error Detail Register, Slave Port n (MPU_EDRn) . . . 449
Figure 208. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0) . . . 451
Figure 209. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1) . . . 451
Figure 210. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2) . . . 452
Figure 211. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3) . . . 454
Figure 212. MPU RGD Alternate Access Control n (MPU_RGDAACn) . . . 455
Figure 213. MPU access evaluation macro . . . 457
Figure 214. Access evaluation macro critical timing path . . . 459
Figure 215. eDMA block diagram . . . 463
Figure 216. eDMA Control Register (EDMA_CR) . . . 467
Figure 217. eDMA Error Status Register (EDMA_ESR) . . . 469
Figure 218. eDMA Enable Request Register (EDMA_ERQRL) . . . 471
Figure 219. eDMA Enable Error Interrupt Register (EDMA_EEIRL) . . . 472
Figure 220. eDMA Set Enable Request Register (EDMA_SERQR) . . . 472
Figure 221. eDMA Clear Enable Request Register (EDMA_CERQR) . . . 473
Figure 222. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR) . . . 473
Figure 223. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR) . . . 474
Figure 224. eDMA Clear Interrupt Request (EDMA_CIRQR) . . . 475
Figure 225. eDMA Clear Error Register (EDMA_CER) . . . 475
Figure 226. eDMA Set START Bit Register (EDMA_SSBR) . . . 476
Figure 227. eDMA Clear DONE Status Bit Register (EDMA_CDSBR) . . . 476
Figure 228. eDMA Interrupt Request Low Register (EDMA_IRQRL) . . . 477
Figure 229. eDMA Error Low Register (EDMA_ERL) . . . 478
Figure 230. EDMA Hardware Request Status Register Low (EDMA_HRSL) . . . 479
Figure 231. eDMA Channel n Priority Register (EDMA_CPRn) . . . 480
Figure 232. TCD structure . . . 482
Figure 233. eDMA operation, part 1 . . . 489
Figure 234. eDMA operation, part 2 . . . 490
Figure 235. eDMA operation, part 3 . . . 491
Figure 236. Example of multiple loop iterations . . . 496
Figure 237. Memory array terms . . . 497
Figure 238. DMA Mux block diagram . . . 505
Figure 239. Channel Configuration Registers (CHCONFIG#*n*) . . . 508
Figure 240. DMA mux triggered channels diagram . . . 511
Figure 241. DMA mux channel triggering: normal operation . . . 511
Figure 242. DMA mux channel triggering: ignored trigger . . . 512
Figure 243. DMA mux channel 4–15 block diagram . . . 513
Figure 244. FlexRay block diagram . . . 518
Figure 245. Module Version Register (MVR) . . . 528
Figure 246. Module Configuration Register (MCR) . . . 529
Figure 247. System Memory Base Address High Register (SYMBADHR) . . . 531
Figure 248. System Memory Base Address Low Register (SYMBADLR) . . . 531
Figure 249. Strobe Signal Control Register (STBSCR) . . . 532
Figure 250. Message Buffer Data Size Register (MBDSR) . . . 535
Figure 251. Message Buffer Segment Size and Utilization Register (MBSSUTR) . . . 535
Figure 252. Protocol Operation Control Register (POCR) . . . 536
Figure 253. Global Interrupt Flag and Enable Register (GIFER) . . . 538
Figure 254. Protocol Interrupt Flag Register 0 (PIFR0) . . . 540
Figure 255. Protocol Interrupt Flag Register 1 (PIFR1) . . . 543
Figure 256. Protocol Interrupt Enable Register 0 (PIER0) . . . 544

Figure 257. Protocol Interrupt Enable Register 1 (PIER1) . . . 545
Figure 258. CHI Error Flag Register (CHIERFR) . . . 546
Figure 259. Message Buffer Interrupt Vector Register (MBIVEC) . . . 549
Figure 260. Channel A Status Error Counter Register (CASERCR) . . . 549
Figure 261. Channel B Status Error Counter Register (CBSERCR) . . . 550
Figure 262. Protocol Status Register 0 (PSR0) . . . 550
Figure 263. Protocol Status Register 1 (PSR1) . . . 552
Figure 264. Protocol Status Register 2 (PSR2) . . . 553
Figure 265. Protocol Status Register 3 (PSR3) . . . 555
Figure 266. Macrotick Counter Register (MTCTR) . . . 556
Figure 267. Cycle Counter Register (CYCTR) . . . 557
Figure 268. Slot Counter Channel A Register (SLTCTAR) . . . 557
Figure 269. Slot Counter Channel B Register (SLTCTBR) . . . 557
Figure 270. Rate Correction Value Register (RTCORVR) . . . 558
Figure 271. Offset Correction Value Register (OFCORVR) . . . 558
Figure 272. Combined Interrupt Flag Register (CIFRR) . . . 559
Figure 273. System Memory Access Time-Out Register (SYMATOR) . . . 560
Figure 274. Sync Frame Counter Register (SFCNTR) . . . 561
Figure 275. Sync Frame Table Offset Register (SFTOR) . . . 561
Figure 276. Sync Frame Table Configuration, Control, Status Register (SFTCCSR) . . . 562
Figure 277. Sync Frame ID Rejection Filter Register (SFIDRFR) . . . 563
Figure 278. Sync Frame ID Acceptance Filter Value Register (SFIDAFVR) . . . 564
Figure 279. Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR) . . . 564
Figure 280. Network Management Vector Registers (NMVR0–NMVR5) . . . 565
Figure 281. Network Management Vector Length Register (NMVLR) . . . 566
Figure 282. Timer Configuration and Control Register (TICCR) . . . 566
Figure 283. Timer 1 Cycle Set Register (TI1CYSR) . . . 567
Figure 284. Timer 1 Macrotick Offset Register (TI1MTOR) . . . 568
Figure 285. Timer 2 Configuration Register 0 (TI2CR0) . . . 568
Figure 286. Timer 2 Configuration Register 1 (TI2CR1) . . . 569
Figure 287. Slot Status Selection Register (SSSR) . . . 569
Figure 288. Slot Status Counter Condition Register (SSCCR) . . . 570
Figure 289. Slot Status Registers (SSR0–SSR7) . . . 572
Figure 290. Slow Status Counter Registers (SSCR0–SSCR3) . . . 574
Figure 291. MTS A Configuration Register (MTSACFR) . . . 574
Figure 292. MTS B Configuration Register (MTSBCFR) . . . 575
Figure 293. Receive Shadow Buffer Index Register (RSBIR) . . . 575
Figure 294. Receive FIFO Selection Register (RFSR) . . . 576
Figure 295. Receive FIFO Start Index Register (RFSIR) . . . 577
Figure 296. Receive FIFO Depth and Size Register (RFDSR) . . . 577
Figure 297. Receive FIFO A Read Index Register (RFARIR) . . . 578
Figure 298. Receive FIFO B Read Index Register (RFBRIR) . . . 578
Figure 299. Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR) . . . 579
Figure 300. Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR) . . . 579
Figure 301. Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR) . . . 579
Figure 302. Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR) . . . 580
Figure 303. Receive FIFO Range Filter Configuration Register (RFRFCFR) . . . 580
Figure 304. Receive FIFO Range Filter Control Register (RFRFCTR) . . . 581
Figure 305. Last Dynamic Slot Channel A Register (LDTXSLAR) . . . 582
Figure 306. Last Dynamic Slot Channel B Register (LDTXSLBR) . . . 582
Figure 307. Protocol Configuration Register 0 (PCR0) . . . 585
Figure 308. Protocol Configuration Register 1 (PCR1) . . . 585

Figure 309. Protocol Configuration Register 2 (PCR2) . . . 585
Figure 310. Protocol Configuration Register 3 (PCR3) . . . 585
Figure 311. Protocol Configuration Register 4 (PCR4) . . . 586
Figure 312. Protocol Configuration Register 5 (PCR5) . . . 586
Figure 313. Protocol Configuration Register 6 (PCR6) . . . 586
Figure 314. Protocol Configuration Register 7 (PCR7) . . . 586
Figure 315. Protocol Configuration Register 8 (PCR8) . . . 586
Figure 316. Protocol Configuration Register 9 (PCR9) . . . 587
Figure 317. Protocol Configuration Register 10 (PCR10) . . . 587
Figure 318. Protocol Configuration Register 11 (PCR11) . . . 587
Figure 319. Protocol Configuration Register 12 (PCR12) . . . 587
Figure 320. Protocol Configuration Register 13 (PCR13) . . . 588
Figure 321. Protocol Configuration Register 14 (PCR14) . . . 588
Figure 322. Protocol Configuration Register 15 (PCR15) . . . 588
Figure 323. Protocol Configuration Register 16 (PCR16) . . . 588
Figure 324. Protocol Configuration Register 17 (PCR17) . . . 588
Figure 325. Protocol Configuration Register 18 (PCR18) . . . 589
Figure 326. Protocol Configuration Register 19 (PCR19) . . . 589
Figure 327. Protocol Configuration Register 20 (PCR20) . . . 589
Figure 328. Protocol Configuration Register 21 (PCR21) . . . 589
Figure 329. Protocol Configuration Register 22 (PCR22) . . . 589
Figure 330. Protocol Configuration Register 23 (PCR23) . . . 590
Figure 331. Protocol Configuration Register 24 (PCR24) . . . 590
Figure 332. Protocol Configuration Register 25 (PCR25) . . . 590
Figure 333. Protocol Configuration Register 26 (PCR26) . . . 590
Figure 334. Protocol Configuration Register 27 (PCR27) . . . 591
Figure 335. Protocol Configuration Register 28 (PCR28) . . . 591
Figure 336. Protocol Configuration Register 29 (PCR29) . . . 591
Figure 337. Protocol Configuration Register 30 (PCR30) . . . 591
Figure 338. Message Buffer Configuration, Control, Status Registers (MBCCSR*n*) . . . 592
Figure 339. Message Buffer Cycle Counter Filter Registers (MBCCFR*n*) . . . 594
Figure 340. Message Buffer Frame ID Registers (MBFIDR*n*) . . . 595
Figure 341. Message Buffer Index Registers (MBIDXR*n*) . . . 595
Figure 342. Physical message buffer structure . . . 596
Figure 343. Individual message buffer structure . . . 598
Figure 344. Receive shadow buffer structure . . . 599
Figure 345. Receive FIFO structure . . . 600
Figure 346. Example of FlexRay memory layout . . . 603
Figure 347. Frame header structure (Receive message buffer and receive FIFO) . . . 604
Figure 348. Frame header structure (Transmit message buffer) . . . 605
Figure 349. Frame header structure (Transmit message buffer for key slot) . . . 605
Figure 350. Receive message buffer slot status structure (ChAB) . . . 608
Figure 351. Receive message buffer slot status structure (ChA) . . . 608
Figure 352. Receive message buffer slot status structure (ChB) . . . 608
Figure 353. Transmit message buffer slot status structure (ChAB) . . . 610
Figure 354. Transmit message buffer slot status structure (ChA) . . . 610
Figure 355. Transmit message buffer slot status structure (ChB) . . . 610
Figure 356. Message buffer data field structure . . . 612
Figure 357. Single transmit message buffer access regions . . . 616
Figure 358. Single transmit message buffer states . . . 617
Figure 359. Message transmission timing . . . 621
Figure 360. Message transmission from HLck state with unlock . . . 621

Figure 361. Null frame transmission from idle state . . . 622
Figure 362. Null frame transmission from HLck state . . . 622
Figure 363. Null frame transmission from HLck state with unlock . . . 622
Figure 364. Null frame transmission from idle state with locking . . . 623
Figure 365. Receive message buffer access regions . . . 624
Figure 366. Receive message buffer states . . . 625
Figure 367. Message reception timing . . . 629
Figure 368. Double transmit buffer structure and data flow . . . 631
Figure 369. Double transmit message buffer access regions layout . . . 631
Figure 370. Double transmit message buffer state diagram (commit side) . . . 633
Figure 371. Double transmit message buffer state diagram (transmit side) . . . 634
Figure 372. Internal message transfer in streaming commit mode . . . 638
Figure 373. Internal message transfer in immediate commit mode . . . 639
Figure 374. Inconsistent channel assignment . . . 641
Figure 375. Message buffer reconfiguration scheme . . . 643
Figure 376. Received frame FIFO filter path . . . 646
Figure 377. Dual channel device mode . . . 648
Figure 378. Single channel device mode (Channel A) . . . 649
Figure 379. Single channel device mode (Channel B) . . . 649
Figure 380. External offset correction write and application timing . . . 650
Figure 381. External rate correction write and application timing . . . 650
Figure 382. Sync table memory layout . . . 651
Figure 383. Sync frame table trigger and generation timing . . . 653
Figure 384. Strobe signal timing (type = pulse, clk_offset = –2) . . . 656
Figure 385. Strobe signal timing (type = pulse, clk_offset = +4) . . . 656
Figure 386. Slot status vector update . . . 658
Figure 387. Slot status counting and SSCR*n* update . . . 659
Figure 388. Scheme of cascaded interrupt request . . . 663
Figure 389. Scheme of combined interrupt flags . . . 664
Figure 390. Transmit data not available . . . 670
Figure 391. Transmit data not available . . . 671
Figure 392. DSPI block diagram . . . 672
Figure 393. DSPI with queues and eDMA . . . 673
Figure 394. DSPI Module Configuration Register (DSPIx_MCR) . . . 679
Figure 395. DSPI Transfer Count Register (DSPIx_TCR) . . . 682
Figure 396. DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn) . . . 683
Figure 397. DSPI Status Register (DSPIx_SR) . . . 688
Figure 398. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER) . . . 690
Figure 399. DSPI PUSH TX FIFO Register (DSPIx_PUSHR) . . . 692
Figure 400. DSPI POP RX FIFO Register (DSPIx_POPR) . . . 694
Figure 401. DSPI Transmit FIFO Register 0–4 (DSPIx_TXFRn) . . . 694
Figure 402. DSPI Receive FIFO Registers 0–4 (DSPIx_RXFRn) . . . 695
Figure 403. SPI serial protocol overview . . . 696
Figure 404. DSPI start and stop state diagram . . . 698
Figure 405. Communications clock prescalers and scalers . . . 701
Figure 406. Peripheral Chip Select strobe timing . . . 703
Figure 407. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8) . . . 705
Figure 408. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8) . . . 706
Figure 409. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$) . . . 708
Figure 410. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$) . . . 709
Figure 411. Example of non-continuous format (CPHA = 1, CONT = 0) . . . 709
Figure 412. Example of continuous transfer (CPHA = 1, CONT = 1) . . . 710

Figure 413. Polarity switching between frames . . . 711
Figure 414. Continuous SCK timing diagram (CONT = 0) . . . 712
Figure 415. Continuous SCK timing diagram (CONT = 1) . . . 712
Figure 416. TX FIFO pointers and counter . . . 718
Figure 417. LIN topology network . . . 722
Figure 418. LINFlex block diagram . . . 722
Figure 419. LINFlex operating modes . . . 724
Figure 420. LINFlex in loop back mode . . . 725
Figure 421. LINFlex in self test mode . . . 726
Figure 422. LIN control register 1 (LINCR1) . . . 727
Figure 423. LIN interrupt enable register (LINIER) . . . 730
Figure 424. LIN status register (LINSR) . . . 732
Figure 425. LIN error status register (LINESR) . . . 735
Figure 426. UART mode control register (UARTCR) . . . 736
Figure 427. UART mode status register (UARTSR) . . . 738
Figure 428. LIN timeout control status register (LINTCSR) . . . 740
Figure 429. LIN output compare register (LINOCR) . . . 741
Figure 430. LIN timeout control register (LINTOCR) . . . 741
Figure 431. LIN fractional baud rate register (LINFBRR) . . . 742
Figure 432. LIN integer baud rate register (LINIBRR) . . . 743
Figure 433. LIN checksum field register (LINCFR) . . . 744
Figure 434. LIN control register 2 (LINCR2) . . . 744
Figure 435. Buffer identifier register (BIDR) . . . 746
Figure 436. Buffer data register LSB (BDRL) . . . 747
Figure 437. Buffer data register MSB (BDRM) . . . 747
Figure 438. Identifier filter enable register (IFER) . . . 748
Figure 439. Identifier filter match index (IFMI) . . . 749
Figure 440. Identifier filter mode register (IFMR) . . . 749
Figure 441. Identifier filter control register (IFCR2*n*) . . . 751
Figure 442. Identifier filter control register (IFCR2*n* + 1) . . . 752
Figure 443. UART mode 8-bit data frame . . . 753
Figure 444. UART mode 9-bit data frame . . . 753
Figure 445. Filter configuration—register organization . . . 760
Figure 446. Identifier match index . . . 761
Figure 447. LIN synch field measurement . . . 762
Figure 448. Header and response timeout . . . 764
Figure 449. FlexCAN block diagram . . . 766
Figure 450. Message buffer structure . . . 772
Figure 451. Rx FIFO structure . . . 776
Figure 452. Module Configuration Register (MCR) . . . 778
Figure 453. Control Register (CTRL) . . . 781
Figure 454. Free Running Timer (TIMER) . . . 785
Figure 455. Rx Global Mask register (RXGMASK) . . . 785
Figure 456. Rx Buffer 14 Mask register (RX14MASK) . . . 786
Figure 457. Rx Buffer 15 Mask register (RX15MASK) . . . 787
Figure 458. Error Counter Register (ECR) . . . 788
Figure 459. Error and Status Register (ESR) . . . 789
Figure 460. Interrupt Masks 1 Register (IMASK1) . . . 791
Figure 461. Interrupt Flags 1 Register (IFLAG1) . . . 792
Figure 462. Rx Individual Mask Registers (RXIMR0–RXIMR31) . . . 793
Figure 463. CAN engine clocking scheme . . . 803
Figure 464. Segments within the bit time . . . 804

Figure 465. Arbitration, match, and move time windows . . . 805
Figure 466. ADC implementation diagram . . . 811
Figure 467. Normal conversion flow . . . 813
Figure 468. Injected sample/conversion sequence . . . 814
Figure 469. Prescaler simplified block diagram . . . 816
Figure 470. Sampling and conversion timings . . . 817
Figure 471. Presampling sequence . . . 819
Figure 472. Presampling sequence with PRECONV = 1 . . . 819
Figure 473. Guarded area . . . 820
Figure 474. Main Configuration Register (MCR) . . . 825
Figure 475. Main Status Register (MSR) . . . 827
Figure 476. Interrupt Status Register (ISR) . . . 828
Figure 477. Interrupt Mask Register (IMR) . . . 829
Figure 478. Watchdog Threshold Interrupt Status Register (WTISR) . . . 830
Figure 479. Watchdog Threshold Interrupt Mask Register (WTIMR) . . . 830
Figure 480. DMA Enable (DMAE) register . . . 832
Figure 481. DMA Channel Select Register 0 (DMAR0) . . . 833
Figure 482. Threshold Control Register (TRC*x*, *x* = [0..3]) . . . 834
Figure 483. Threshold Register (THRHLR[0:3]) . . . 835
Figure 484. Presampling Control Register (PSCR) . . . 835
Figure 485. Presampling Register 0 (PSR0) . . . 836
Figure 486. Conversion Timing Registers CTR[0] . . . 837
Figure 487. Normal Conversion Mask Register 0 (NCMR0) . . . 838
Figure 488. Injected Conversion Mask Register 0 (JCMR0) . . . 838
Figure 489. Power-Down Exit Delay Register (PDEDR) . . . 839
Figure 490. Channel Data Registers (CDR[0..26]) . . . 840
Figure 491. Cross triggering unit diagram . . . 842
Figure 492. TGS in triggered mode . . . 843
Figure 493. Example timing for TGS in triggered mode . . . 844
Figure 494. TGS in sequential mode . . . 845
Figure 495. Example timing for TGS in sequential mode . . . 845
Figure 496. TGS counter cases . . . 846
Figure 497. Scheduler subunit . . . 847
Figure 498. Reload error scenario . . . 851
Figure 499. Trigger Generator Sub-unit Input Selection Register (TGSISR) . . . 859
Figure 500. Trigger Generator Sub-unit Control Register (TGSCR) . . . 861
Figure 501. Trigger x Compare Register (TxCR, *x* = 0...7) . . . 862
Figure 502. TGS Counter Compare Register (TGSCCR) . . . 862
Figure 503. TGS Counter Reload Register (TGSCRR) . . . 863
Figure 504. Commands list control register 1 (CLCR1) . . . 863
Figure 505. Commands list control register 2 (CLCR2) . . . 864
Figure 506. Trigger handler control register 1 (THCR1) . . . 864
Figure 507. Trigger handler control register 2 (THCR2) . . . 866
Figure 508. Commands list register *x* (*x* = 1,...,24) (CMS = 0) . . . 868
Figure 509. Commands list register x (x = 1,...,24) (CMS = 1) . . . 869
Figure 510. FIFO DMA control register (FDCR) . . . 869
Figure 511. FIFO control register (FCR) . . . 870
Figure 512. FIFO threshold register (FTH) . . . 872
Figure 513. FIFO status register (FST) . . . 872
Figure 514. FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*) . . . 874
Figure 515. FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*) . . . 875
Figure 516. Cross triggering unit error flag register (CTUEFR) . . . 875

Figure 517. Cross triggering unit interrupt flag register (CTUIFR) . . . 876
Figure 518. Cross triggering unit interrupt/DMA register (CTUIR) . . . 877
Figure 519. Control ON time register (COTR) . . . 878
Figure 520. Cross triggering unit control register (CTUCR) . . . 879
Figure 521. Cross triggering unit digital filter (CTUDF) . . . 880
Figure 522. Cross triggering unit power control register (CTUPCR) . . . 880
Figure 523. SEMA4 block diagram . . . 882
Figure 524. SEMA4 gate *n* register (SEMA4_GATE*n*) . . . 885
Figure 525. Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE) . . . 885
Figure 526. Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF) . . . 886
Figure 527. Semaphores (secure) reset gate *n* (SEMA4_RSTGT) . . . 887
Figure 528. Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF) . . . 889
Figure 529. eTimer block diagram . . . 896
Figure 530. eTimer channel block diagram . . . 897
Figure 531. Compare register 1 (COMP1) . . . 901
Figure 532. Compare register 2 (COMP2) . . . 902
Figure 533. Capture register 1 (CAPT1) . . . 902
Figure 534. Capture register 2 (CAPT2) . . . 903
Figure 535. Load register (LOAD) . . . 903
Figure 536. Hold register (HOLD) . . . 904
Figure 537. Counter register (CNTR) . . . 904
Figure 538. Control register 1 (CTRL1) . . . 905
Figure 539. Control register 2 (CTRL2) . . . 907
Figure 540. Control register 3 (CTRL3) . . . 909
Figure 541. Status register (STS) . . . 910
Figure 542. Interrupt and DMA enable register (INTDMA) . . . 912
Figure 543. Comparator Load 1 (CMPLD1) . . . 913
Figure 544. Comparator Load 2 (CMPLD2) . . . 913
Figure 545. Compare and Capture Control register (CCCTRL) . . . 914
Figure 546. Input Filter register (FILT) . . . 916
Figure 547. Watchdog Time-out Low Word register (WDTOL) . . . 917
Figure 548. Watchdog Time-Out High Word register (WDTOH) . . . 917
Figure 549. Channel Enable register (ENBL) . . . 917
Figure 550. DMA Request 0 Select register (DREQ0) . . . 918
Figure 551. DMA Request 1 Select register (DREQ1) . . . 918
Figure 552. Quadrature incremental position encoder . . . 921
Figure 553. Triggered Count mode (length = 1) . . . 922
Figure 554. One-Shot mode (length = 1) . . . 922
Figure 555. Pulse Output mode . . . 923
Figure 556. Register protection module block diagram . . . 928
Figure 557. Register protection memory diagram . . . 929
Figure 558. Soft Lock Bit Register (SLBR*n*) . . . 931
Figure 559. Global Configuration Register (GCR) . . . 932
Figure 560. Change lock settings directly via area #4 . . . 933
Figure 561. Change lock settings for 16-bit protected addresses . . . 934
Figure 562. Change lock settings for 32-bit protected addresses . . . 934
Figure 563. Change lock settings for mixed protection . . . 935
Figure 564. Enable locking via mirror module space (area #3) . . . 935
Figure 565. Enable locking for protected and unprotected addresses . . . 935
Figure 566. SWT Control Register (SWT_CR) . . . 938
Figure 567. SWT Interrupt Register (SWT_IR) . . . 940
Figure 568. SWT Time-Out register (SWT_TO) . . . 940

Figure 569. SWT Window register (SWT_WN) . . . 941
Figure 570. SWT Service Register (SWT_SR) . . . 942
Figure 571. SWT Counter Output register (SWT_CO) . . . 942
Figure 572. SWT Service Register (SWT_SR) . . . 943
Figure 573. Pseudorandom Key Generator . . . 944
Figure 574. FCCU top-level diagram . . . 948
Figure 575. FCCU Control Register (FCCU_CTRL) . . . 952
Figure 576. FCCU CTRL Key Register (FCCU_CTRLK) . . . 954
Figure 577. FCCU Configuration Register (FCCU_CFG) . . . 955
Figure 578. FCCU CF Configuration Register (FCCU_CF_CFG0..3) . . . 957
Figure 579. FCCU NCF Configuration Register (FCCU_NCF_CFG0..3) . . . 958
Figure 580. FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0) . . . 959
Figure 581. FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1) . . . 959
Figure 582. FCCU CFS Configuration Register 2..7 (FCCU_CFS_CFG2..7) . . . 960
Figure 583. FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7) . . . 961
Figure 584. FCCU CF Status Register (FCCU_CFS0..3) . . . 962
Figure 585. FCCU CF Key Register (FCCU_CFK) . . . 963
Figure 586. FCCU NCF Status Register (FCCU_NCFS0..3) . . . 965
Figure 587. FCCU NCF Key Register (FCCU_NCFK) . . . 966
Figure 588. FCCU NCF Enable Register (FCCU_NCFE0..3) . . . 966
Figure 589. FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3) . . . 967
Figure 590. FCCU NCF Time-out Register (FCCU_NCF_TO) . . . 968
Figure 591. FCCU CFG Timeout Register (FCCU_CFG_TO) . . . 969
Figure 592. FCCU Status Register (FCCU_STAT) . . . 970
Figure 593. FCCU CF Fake Register (FCCU_CFF) . . . 971
Figure 594. FCCU NCF Fake Register (FCCU_NCFF) . . . 972
Figure 595. FCCU IRQ Status Register (FCCU_IRQ_STAT) . . . 973
Figure 596. FCCU IRQ Enable Register (FCCU_IRQ_EN) . . . 974
Figure 597. FCCU XTMR Register (FCCU_XTMR) . . . 975
Figure 598. FCCU MCS Register (FCCU_MCS) . . . 975
Figure 599. FCCU state diagram . . . 978
Figure 600. Self-checking reaction . . . 979
Figure 601. Critical FAULT recovery (a) . . . 980
Figure 602. Critical FAULT recovery (b) . . . 981
Figure 603. Non-critical FAULT (ALARM state) recovery (a) . . . 981
Figure 604. Non-critical FAULT (ALARM state) recovery (b) . . . 982
Figure 605. Non-critical FAULT (ALARM -> FAULT state) recovery . . . 982
Figure 606. Critical FAULT (nesting) recovery . . . 983
Figure 607. Critical FAULT (and non-critical FAULT nesting) recovery . . . 984
Figure 608. NVM interface . . . 985
Figure 609. Dual-rail protocol (slow switching mode) . . . 987
Figure 610. Dual-rail protocol (fast switching mode) . . . 987
Figure 611. Time-switching protocol . . . 988
Figure 612. Bi-stable protocol . . . 989
Figure 613. NMI Status Flag Register (NSR) . . . 992
Figure 614. NMI Configuration Register (NCR) . . . 993
Figure 615. NMI pad diagram . . . 994
Figure 616. PIT block diagram . . . 996
Figure 617. PIT Module Control Register (PITMCR) . . . 998
Figure 618. Timer Load Value Register *n* (LDVAL*n*) . . . 999
Figure 619. Current Timer Value register *n* (CVAL*n*) . . . 1000
Figure 620. Timer Control register *n* (TCTRL*n*) . . . 1001

Figure 621. Timer Flag register *n* (TFLG*n*) . . . 1002
Figure 622. Stopping and starting a timer . . . 1003
Figure 623. Modifying running timer period . . . 1003
Figure 624. Dynamically setting a new load value . . . 1003
Figure 625. STM Control Register (STM_CR) . . . 1007
Figure 626. STM Count Register (STM_CNT) . . . 1007
Figure 627. STM Channel Control Register (STM_CCR*n*) . . . 1008
Figure 628. STM Channel Interrupt Register (STM_CIR*n*) . . . 1009
Figure 629. STM Channel Compare Register (STM_CMP*n*) . . . 1009
Figure 630. CRC top level diagram . . . 1012
Figure 631. CRC-CCITT engine concept scheme . . . 1013
Figure 632. CRC computation flow . . . 1014
Figure 633. CRC Configuration Register (CRC_CFG) . . . 1015
Figure 634. CRC Input Register (CRC_INP) . . . 1016
Figure 635. CRC Current Status Register (CRC_CSTAT) . . . 1017
Figure 636. CRC Output Register (CRC_OUTP) . . . 1018
Figure 637. CRC Output Check Register (CRC_OUTP_CHK) . . . 1018
Figure 638. DMA-CRC Transmission Sequence . . . 1020
Figure 639. DMA-CRC Reception Sequence . . . 1022
Figure 640. Boot mode selection . . . 1024
Figure 641. Reset Configuration Half Word (RCHW) . . . 1026
Figure 642. SPC56xP60/54 Flash partitioning and RCHW search . . . 1027
Figure 643. BAM logic flow . . . 1029
Figure 644. Password check flow . . . 1033
Figure 645. Start address, VLE bit and download size in bytes . . . 1034
Figure 646. LINFlex bit timing in UART mode . . . 1035
Figure 647. FlexCAN bit timing . . . 1036
Figure 648. BAM Autoscan code flow . . . 1039
Figure 649. Baud measurement on UART boot . . . 1039
Figure 650. BAM rate measurement flow during UART boot . . . 1040
Figure 651. Baud rate deviation between host and SPC56xP60/54 . . . 1042
Figure 652. Bit time measure . . . 1043
Figure 653. BAM rate measurement flow during FlexCAN boot . . . 1044
Figure 654. Censorship control in flash memory boot mode . . . 1051
Figure 655. Voltage Regulator Control register (VREG_CTL) . . . 1055
Figure 656. Voltage Regulator Status register (VREG_STATUS) . . . 1056
Figure 657. JTAG controller block diagram . . . 1058
Figure 658. 5-bit Instruction register . . . 1061
Figure 659. Device identification register . . . 1061
Figure 660. Shifting data through a register . . . 1063
Figure 661. IEEE 1149.1-2001 TAP controller finite state machine . . . 1064
Figure 662. e200z0 OnCE block diagram . . . 1068
Figure 663. OnCE Command register (OCMD) . . . 1069
Figure 664. Nexus Port Controller Block Diagram . . . 1072
Figure 665. 4-Bit Instruction Register . . . 1076
Figure 666. Nexus Device ID Register . . . 1077
Figure 667. Port Configuration Register (PCR) . . . 1078
Figure 668. MSEO Transfers (for 2-bit MSEO) . . . 1081
Figure 669. Message Field Sizes . . . 1082
Figure 670. Transmission Sequence of Messages . . . 1083
Figure 671. Shifting Data Into Register . . . 1083
Figure 672. IEEE 1149.1-2001 TAP Controller State Machine . . . 1085

Figure 673. NEXUS Controller State Machine 1086
Figure 674. IEEE 1149.1 Controller Command Input 1087

Preface

Overview

The primary objective of this document is to define the functionality of the SPC56xP60/54 family of microcontrollers for use by software and hardware developers. The SPC56xP60/54 family is built on Power Architecture® technology and integrates technologies that are important for today's electrical hydraulic power steering (EHPS), electric power steering (EPS), and airbag applications.

As with any technical documentation, it is the reader's responsibility to be sure he or she is using the most recent version of the documentation.

To locate any published errata or updates for this document, visit the ST Web site at www.st.com.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC56xP60/54 device. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

Chapter organization and device-specific information

This document includes chapters that describe:

- The device as a whole
- The functionality of the individual modules on the device

In the latter, any device-specific information is presented in the section "Information Specific to This Device" at the beginning of the chapter.

References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC56xP60/54:

- IEEE-ISTO 5001™ - 2003 and 2010, The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface
- IEEE 1149.1-2001 standard - IEEE Standard Test Access Port and Boundary-Scan Architecture

1 Introduction

1.1 The SPC56xP60/54 microcontroller family

The SPC56xP60/54 microcontroller is built on the Power Architecture® platform and targets chassis market segment, specifically the airbag application space. The Power Architecture based 32-bit microcontrollers represent the latest achievement in integrated automotive application controllers.

SPC56xP60/54 devices are built around a dual-core platform for optimized performance-power consumption trade-off but is available also as single-core version.

The core selected for the device is the Harvard bus interface version of the e200z0h to cover the low-end chassis application space.

The e200 processor family is a set of CPU cores that implement low-cost versions of the Power Architecture technology. The e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance. The e200z0h processor integrates an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable to sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by branch unit to allow single-cycle branches in some cases. The e200z0h core is a single-issue, 32-bit Power Architecture technology VLE only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). Instead of the base Power Architecture instruction set support, the e200z0h core only implements the VLE (variable length encoding) APU, providing improved code density.

The SPC56xP60/54 has a single level of memory hierarchy consisting of 80 KB on-chip SRAM and 1088 KB (1024 KB program + 64 KB data) of on-chip flash memory. Both the SRAM and the flash memory can hold instructions and data.

The timer functions of SPC56xP60/54 are performed by the eTimer Modular Timer System. The two eTimer modules implement enhanced timer features (six channels each for a total of 12) including dedicated motor control quadrature decode functionality and DMA support.

Off-chip communication is performed by a suite of serial protocols including FlexRay, CANs, enhanced SPIs (DSPI), and SCIs (LINFlex).

The System Integration Unit Lite (SIUL) performs several chip-wide configuration functions. Pad configuration and general-purpose input/output (GPIO) are controlled from the SIUL. External interrupts and reset control are also found in the SIUL. The internal multiplexer sub-block (IOMUX) provides multiplexing of daisy chaining the DSPIs and external interrupt signal.

As the SPC56xP60/54 is built on a wider legacy of Power Architecture-based devices, when applicable and possible, reusing or enhancement of existing IP, design and concepts is adopted.

1.2 Application examples

Figure 1 outlines a typical airbag application built around the SPC56xP60/54 microcontroller.

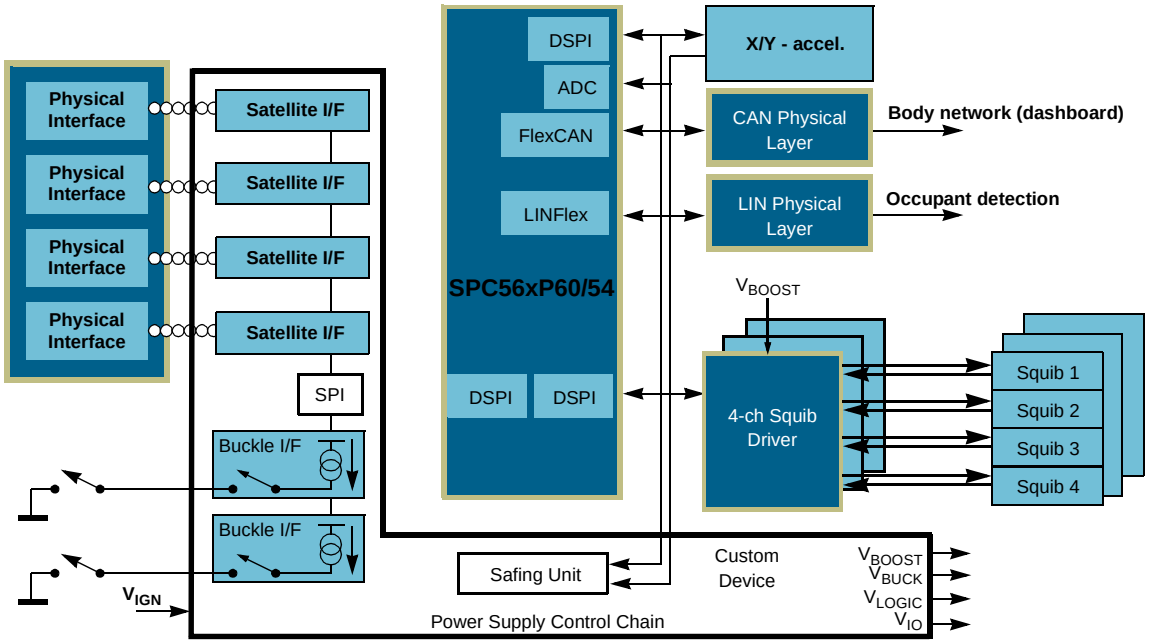


Figure 1. Airbag application

1.3 SPC56xP60/54 Device Summary

Table 1 provides a summary of different members of the SPC56xP60/54 family and their features—relative to Enhanced Full-featured version—to enable a comparison among the family members and an understanding of the range of functionality offered within this family.

Table 1. SPC56xP60/54 device comparison

Feature	SPC560P54	SPC560P60	SPC56AP54	SPC56AP60
Code Flash memory (with ECC)	768 KB	1 MB	768 KB	1 MB
Data Flash / EE (with ECC)	64 KB			
SRAM (with ECC)	64 KB	80 KB	64 KB	80 KB
Processor core	32-bit e200z0h		32-bit Dual e200z0h	
Instruction set	VLE			
CPU performance	0-64 MHz			
FMPLL (frequency-modulated phase-locked loop) modules	1			
INTC (interrupt controller) channels	148			
PIT (periodic interrupt timer)	1 (includes four 32-bit timers)			
Enhanced DMA (direct memory access) channels	16			

Table 1. SPC56xP60/54 device comparison (continued)

Feature		SPC560P54	SPC560P60	SPC56AP54	SPC56AP60
FlexRay		Yes (64 message buffer)			
FlexCAN (controller area network)		3(1),(2)			
Safety port		Yes (via third FlexCAN module)			
FCCU (fault collection and control unit)		Yes(3)			
CTU (cross triggering unit)		Yes			
eTimer channels		2 × 6			
FlexPWM (pulse-width modulation) channels		No			
Analog-to-digital converters (ADC)		One (10-bit, 27-channel)(4)			
LINFlex modules		2 (1 × Master/Slave, 1 × Master only)			
DSPI (deserial serial peripheral interface) modules		5(5)			
CRC (cyclic redundancy check) units		2(6)			
JTAG interface		Yes			
Nexus port controller (NPC)		Yes (Level 2+)(7)			
Supply	Digital power supply(8)	3.3 V or 5 V single supply with external transistor			
	Analog power supply	3.3 V or 5 V			
	Internal RC oscillator	16 MHz			
	External crystal oscillator	4–40 MHz			
Packages		LQFP100 LQFP144			LQFP100 LQFP144 LQFP176 (9)
Temperature	Standard ambient temperature	–40 to 125 °C			

1. Each FlexCAN module has 32 message buffers.
2. One FlexCAN module can act as a Safety Port with a bit rate as high as 7.5 Mbit/s.
3. Enhanced FCCU version
4. Same amount of ADC channels as on SPC560P44/50 not considering the internally connected ones. 26 channels on LQFP144 and 16 channels on LQFP100.
5. Increased number of CS for DSPI_1
6. Upgraded specification with addition of 8-bits polynomial (CRC-8 VDA CAN) support and 3rd context
7. Improved debugging capability with data trace capability and increased Nexus throughput available on emulation package
8. 3.3 V range and 5 V range correspond to different orderable parts.
9. Software development package only. Not available for production.

SPC56xP60/54 is present on the market in three different options enabling different features: Enhanced Full-featured, Full-featured, and Airbag configuration. *Table 2* shows the main differences between the three versions.

Table 2. SPC56xP60/54 device configuration difference

Feature	Enhanced Full-featured	Full-featured	Airbag
FlexCAN (controller area network)	3	2	2
CTU (cross triggering unit)	Yes		No
FlexRay	Yes (64 message buffer)		No
DSPI (deserial serial peripheral interface) modules	5		4
CRC (cyclic redundancy check) unit	2		1

1.4 Device block diagram

Figure 2 shows a top-level block diagram of the SPC56xP60/54 MCU. *Table 3* summarizes the functions of the blocks.

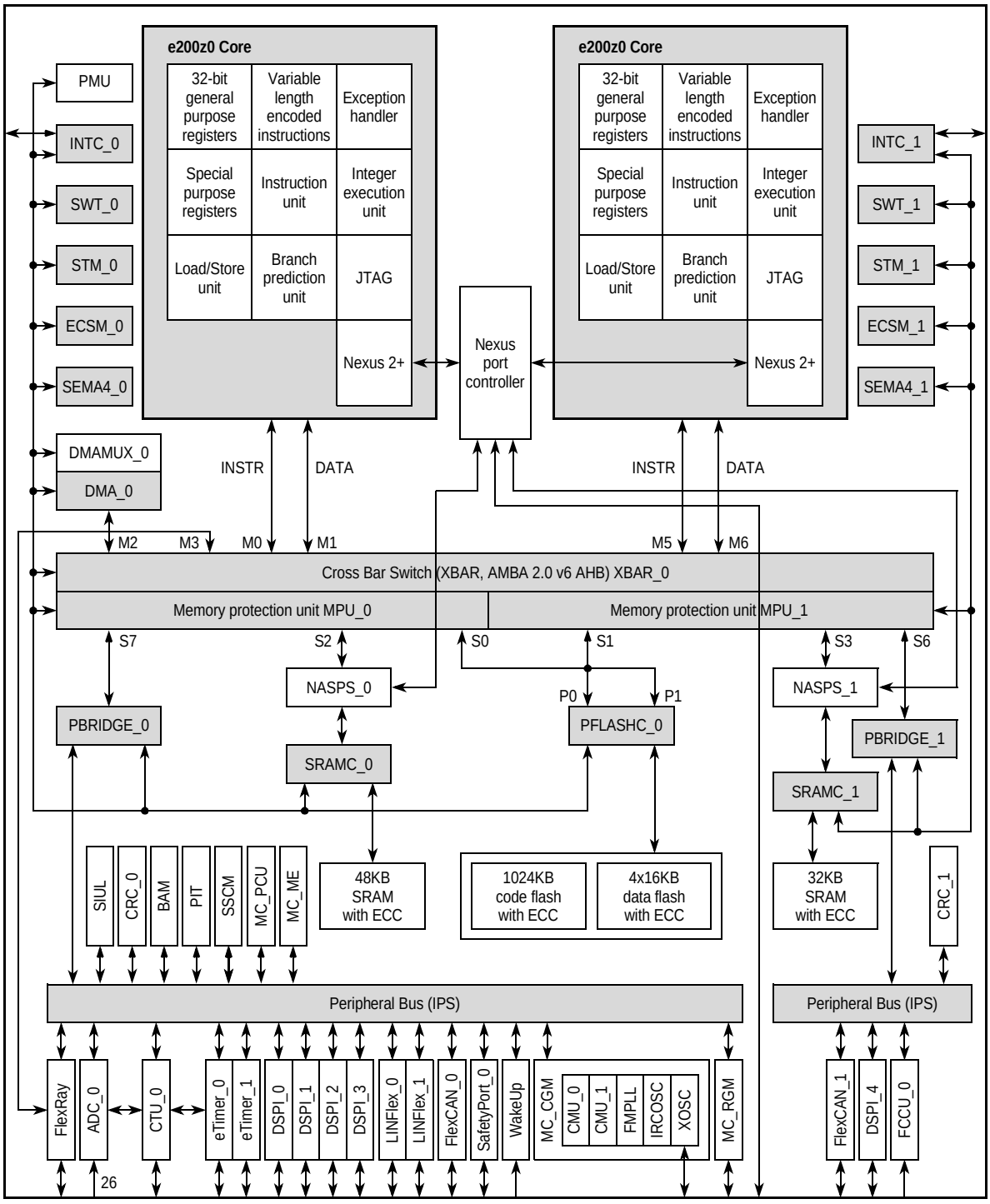


Figure 2. SPC56xP60/54 block diagram

Table 3. SPC56xP60/54 series block summary

Block	Function
Analog-to-digital converter (ADC)	Multi-channel, 10-bit analog-to-digital converter
Boot assist module (BAM)	Block of read-only memory containing VLE code which is executed according to the boot mode of the device
Clock generation module (MC_CGM)	Provides logic and control required for the generation of system and peripheral clocks
Controller area network (FlexCAN)	Supports the standard CAN communications protocol
Cross triggering unit (CTU)	Enables synchronization of ADC conversions with a timer event from the eMIOS or from the PIT
Crossbar switch (XBAR)	Supports simultaneous connections between two master ports and three slave ports. The crossbar supports a 32-bit address bus width and a 32-bit data bus width.
Cyclic redundancy checker (CRC) unit	Is dedicated to the computation of CRC off-loading the CPU. Each context has a separate CRC computation engine in order to allow the concurrent computation of the CRC of multiple data streams.
Deserial serial peripheral interface (DSPI)	Provides a synchronous serial interface for communication with external devices
Enhanced direct memory access (eDMA)	Performs complex data transfers with minimal intervention from a host processor via "*n*" programmable channels
Enhanced timer (eTimer)	Provides enhanced programmable up/down modulo counting
Error correction status module (ECSM)	Provides a myriad of miscellaneous control functions for the device including program-visible information about configuration and revision levels, a reset status register, wakeup control for exiting sleep modes, and optional features such as information on memory errors reported by error-correcting codes
External oscillator (XOSC)	Provides an output clock used as input reference for FMPLL_0 or as reference clock for specific modules depending on system needs
Fault collection and control unit (FCCU)	Provides functional safety to the device
Flash memory	Provides non-volatile storage for program code, constants and variables
FlexRay (FlexRay communication controller)	Provides high-speed distributed control for advanced automotive applications
Frequency-modulated phase-locked loop (FMPLL)	Generates high-speed system clocks and supports programmable frequency modulation
Interrupt controller (INTC)	Provides priority-based preemptive scheduling of interrupt requests
JTAG controller	Provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode
LINFlex controller	Manages a high number of LIN (Local Interconnect Network protocol) messages efficiently with a minimum of CPU load
Mode entry module (MC_ME)	Provides a mechanism for controlling the device operational mode and mode transition sequences in all functional states; also manages the power control unit, reset generation module and clock generation module, and holds the configuration, control and status registers accessible for applications

Table 3. SPC56xP60/54 series block summary (continued)

Block	Function
Peripheral bridge (PBRIDGE)	Is the interface between the system bus and on-chip peripherals
Periodic interrupt timer (PIT)	Produces periodic interrupts and triggers
Power control unit (MC_PCU)	Reduces the overall power consumption by disconnecting parts of the device from the power supply via a power switching device; device components are grouped into sections called “power domains” which are controlled by the PCU
Reset generation module (MC_RGM)	Centralizes reset sources and manages the device reset sequence of the device
Semaphore unit (SEMA4)	Provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access
Static random-access memory (SRAM)	Provides storage for program code, constants, and variables
System integration unit lite (SIUL)	Provides control over all the electrical pad controls and up 32 ports with 16 bits of bidirectional, general-purpose input and output signals and supports up to 32 external interrupts with trigger event configuration
System status and configuration module (SSCM)	Provides system configuration and status data (such as memory size and status, device mode and security status), device identification data, debug status port enable and selection, and bus and peripheral abort enable/disable
System timer module (STM)	Provides a set of output compare events to support AUTOSAR(1) and operating system tasks
System watchdog timer (SWT)	Provides protection from runaway code
Wakeup unit (WKPU)	Supports up to 18 external sources that can generate interrupts or wakeup events, of which 1 can cause non-maskable interrupt requests or wakeup events.

1. AUTOSAR: AUTomotive Open System ARchitecture (see www.autosar.org)

1.5 Critical performance parameters

- Fully static operation, 0 – 64 MHz
- –40 °C to 150 °C junction temperature
- Low power design
  - Dynamic clock gating of core and peripherals
  - Software controlled clock gating of peripherals
  - HALT and STOP mode available for power reduction
- Fabricated in 90 nm process
- 1.2 V nominal internal logic
- Nexus pins operate at V_{DDIO} (no dedicated power supply)
  - Unused pins configurable as GPIO
- 10-bit ADC conversion time < 1 µs
- Internal voltage regulator (VREG) with external ballast transistor enables control with a single input rail
  - 3.0 V–3.6 V or 4.5 V–5.5 V input supply voltage
- Configurable pins
  - Selectable slew rate for EMI reduction
  - Selectable pull-up, pull-down, or no pull on all pins
  - Selectable open drain
  - Support for 3.3 V or 5 V I/O levels

1.6 Feature summary

On-chip modules available within the family include the following features:

- 64 MHz, dual issue, 32-bit CPU core complex (e200z0h)
  - Compliant with Power Architecture® embedded category
  - Variable Length Encoding (VLE)
- Memory organizazion
  - Up to 1024 KB on-chip code Flash memory with additional 64 KB for EEPROM emulation (data flash), with ECC, with erase/program controller
  - Up to 80 KB on-chip SRAM with ECC
- Fail safe protection
  - ECC protection on system SRAM and Flash
  - Safety port
  - SWT with servicing sequence pseudo-random generator
  - Power management
  - Non-maskable interrupt for both cores
  - Fault collection and control unit (FCCU)
  - Safe mode of system-on-chip (SoC)
  - Register protection scheme
- Nexus® L2+ interface
- Single 3.3 V or 5 V supply for I/Os and ADC
- 2 on-platform peripherals set with 2 INTC
- 16-channel eDMA controller with multiple transfer request sources
- General purpose I/Os (80 GPIO + 26 GPI on LQFP144; 49 GPIO + 16 GPI on LQFP100)
- 2 general purpose eTimer units
  - 6 timers, each with up/down count capabilities
  - 16-bit resolution, cascadable counters
  - Quadrature decode with rotation direction flag
  - Double buffer input capture and output compare
- Communications interfaces
  - 2 LINFlex modules (LIN 2.1)
  - 5 DSPI modules with automatic chip select generation
  - 2 FlexCAN interfaces (2.0B Active) with 32 message buffers
  - 1 Safety port based on FlexCAN; usable as third CAN when not used as safety port
  - 1 FlexRay™ module (V2.1) with dual or single channel, 64 message buffers and up to 10 Mbit/s
- 2 CRC units with three contexts and 3 hardwired polynomials(CRC8,CRC32 and CRC-16-CCITT)
- 10-bit A/D converter
  - 27 input channels and pre-sampling feature
  - Conversion time < 1 µs including sampling time at full precision

  - Programmable cross triggering unit (CTU)
  - 4 analog watchdog with interrupt capability
- On-chip CAN/UART Bootstrap loader with boot assist module (BAM)
  - Ambient temperature ranges: –40 to 125 °C or –40 to 105 °C

1.7 Features details

1.7.1 High performance e200z0h core processor

The e200z0h Power Architecture core provides the following features:

- High performance e200z0 core processor for managing peripherals and interrupts
- Single issue 4-stage pipeline in-order execution 32-bit Power Architecture CPU
- Harvard architecture
- Variable length encoding (VLE), allowing mixed 16-bit and 32-bit instructions
  - Results in smaller code size footprint
  - Minimizes impact on performance
- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1-cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- Thirty-two 32-bit general purpose registers (GPRs)
- Separate instruction bus and load/store bus Harvard architecture
- Hardware vectored interrupt support
- Reservation instructions for implementing read-modify-write constructs
- Long cycle time instructions, except for guarded loads, do not increase interrupt latency
- Extensive system development support through Nexus debug port
- Non maskable Interrupt support

1.7.2 Crossbar switch (XBAR)

The XBAR multi-port crossbar switch supports simultaneous connections between six master ports and six slave ports. The crossbar supports a 32-bit address bus width and a 32-bit data bus width.

The crossbar allows for two concurrent transactions to occur from any master port to any slave port; but one of those transfers must be an instruction fetch from internal flash memory. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions. Requesting masters are treated with equal priority and will be granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access.

The crossbar provides the following features:

- 6 master ports:
  - 2 e200z0 core complex Instruction ports
  - 2 e200z0 core complex Load/Store Data ports
  - eDMA
  - FlexRay
- 6 slave ports:
  - 2 Flash memory (code flash and data flash)
  - 2 SRAM (48 KB + 32 KB)
  - 2 PBRIDGE
- 32-bit internal address, 32-bit internal data paths
- Fixed Priority Arbitration based on Port Master
- Temporary dynamic priority elevation of masters

1.7.3 Enhanced direct memory access (eDMA)

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 16 programmable channels, with minimal intervention from the host processor. The hardware micro architecture includes a DMA engine which performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation is utilized to minimize the overall block size.

The eDMA module provides the following features:

- 16 channels support independent 8, 16 or 32-bit single value or block transfers
- Supports variable sized queues and circular queues
- Source and destination address registers are independently configured to post-increment or remain constant
- Each transfer is initiated by a peripheral, CPU, or eDMA channel request
- Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer
- DMA transfers possible between system memories, DSPIs, ADC, eTimer and CTU
- Programmable DMA Channel Multiplexer for assignment of any DMA source to any available DMA channel with up to 30 potential request sources
- eDMA abort operation through software

1.7.4 On-chip flash memory with ECC

The SPC56xP60/54 provides up to 1024 KB of programmable, non-volatile, flash memory. The non-volatile memory (NVM) can be used for instruction and/or data storage. The flash memory module interfaces the system bus to a dedicated flash memory array controller. It supports a 32-bit data bus width at the system bus port, and a 128-bit read data interface to flash memory. The module contains a four-entry, 4x128-bit prefetch buffers. Prefetch buffer hits allow no-wait responses. Normal flash memory array accesses are registered and are forwarded to the system bus on the following cycle, incurring 2 wait states.

The flash memory module provides the following features:

- Up to 1024 KB flash memory
  - 14 blocks (2×16 KB + 2×32 KB + 2×16 KB + 2×64 KB + 6×128 KB) code flash
  - 4 blocks (16 KB + 16 KB + 16 KB + 16 KB) data flash
  - Full Read While Write (RWW) capability between code and data flash
- Four 128-bit wide prefetch buffers to provide single cycle in-line accesses (prefetch buffers can be configured to prefetch code or data or both)
- Typical flash memory access time: 0 wait states for buffer hits, 2 wait states for page buffer miss at 64 MHz
- Hardware managed flash memory writes handled by 32-bit RISC Krypton engine
- Hardware and software configurable read and write access protections on a per-master basis.
- Configurable access timing allowing use in a wide range of system frequencies.
- Multiple-mapping support and mapping-based block access timing (0–31 additional cycles) allowing use for emulation of other memory types.
- Software programmable block program/erase restriction control.
- Erase of selected block(s)
- Read page size of 128 bits (4 words)
- 64-bit ECC with single-bit correction, double-bit detection for data integrity
- Embedded hardware program and erase algorithm
- Erase suspend, program suspend and erase-suspended program
- Censorship protection scheme to prevent flash memory content visibility
- Hardware support for EEPROM emulation

1.7.5 On-chip SRAM with ECC

The SPC56xP60/54 SRAM module provides a general-purpose memory of up to 80 KB.

The SRAM module provides the following features:

- Supports read/write accesses mapped to the SRAM memory from any master
- Up to 80 KB general purpose RAM
  - 2 blocks (48 KB + 32 KB)
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory
- Typical SRAM access time: 0 wait state for reads and 32-bit writes; 1 wait state for 8- and 16-bit writes if back to back with a read to same memory block

1.7.6 Interrupt controller (INTC)

The INTC (interrupt controller) provides priority-based preemptive scheduling of interrupt requests, suitable for statically scheduled hard real-time systems.

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is executing the interrupt service routine (ISR) has been minimized. The INTC provides a unique vector for each interrupt request source for quick determination of which ISR needs to be executed. It also provides an ample number of priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. To

allow the appropriate priorities for each source of interrupt request, the priority of each interrupt request is software configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource can not preempt each other.

The INTC provides the following features:

- Unique 9-bit vector for each separate interrupt source
- 8 software triggerable interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Ability to modify the ISR or task priority.
  - Modifying the priority can be used to implement the Priority Ceiling Protocol for accessing shared resources.
- 2 external high priority interrupts directly accessing the main core and IOP critical interrupt mechanism

The INTC module is replicated for each processor.

1.7.7 System clocks and clock generation

The following list summarizes the system clock and clock generation on the SPC56xP60/54:

- Lock detect circuitry continuously monitors lock status
- Loss of clock (LOC) detection for PLL outputs
- Programmable output clock divider (÷1, ÷2, ÷4, ÷8)
- Programmable output clock divider (÷1, ÷2, ÷3 to ÷256)
- eTimer module running at the same frequency as the e200z0h core
- On-chip oscillator with automatic level control
- Internal 16 MHz RC oscillator for rapid start-up and safe mode
  - Supports frequency trimming by user application

1.7.8 Frequency modulated phase-locked loop (FMPLL)

The FMPLL allows the user to generate high speed system clocks from a 4 MHz to 40 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor, output clock divider ratio are all software configurable.

The FMPLL has the following major features:

- Input clock frequency from 4 MHz to 40 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the PLL to relock
- Modulation enabled/disabled through software
- Triangle wave modulation
- Programmable modulation depth (±0.25% to ±4% deviation from center frequency)
  - Programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation

1.7.9 Main oscillator

The main oscillator provides these features:

- Input frequency range 4 MHz to 40 MHz
- Crystal input mode or Oscillator input mode
- PLL reference

1.7.10 Internal RC oscillator

This device has an RC ladder phase-shift oscillator. The architecture uses constant current charging of a capacitor. The voltage at the capacitor is compared by the stable bandgap reference voltage.

The RC Oscillator provides these features:

- Nominal frequency 16 MHz
- ±5% variation over voltage and temperature after process trim
- Clock output of the RC oscillator serves as system clock source in case loss of lock or loss of clock is detected by the PLL
- RC oscillator is used as the default system clock during startup

1.7.11 Periodic interrupt timer (PIT)

The PIT module implements these features:

- Up to four general purpose interrupt timers
- 32-bit counter resolution
- Clocked by system clock frequency
- Each channel can be used as trigger for a DMA request

1.7.12 System timer module (STM)

The STM module implements these features:

- 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

The STM module is replicated for each processor.

1.7.13 Software watchdog timer (SWT)

The SWT has the following features:

- Fault tolerant output
- Safe internal RC oscillator as reference clock
- Windowed watchdog
- Program flow control monitor with 16-bit pseudorandom key generation

The SWT module is replicated for each processor.

1.7.14 Fault collection and control unit (FCCU)

The FCCU provides an independent fault reporting mechanism even if the CPU is exhibiting unstable behaviors. The FCCU module has the following features:

- Redundant collection of hardware checker results
- Redundant collection of error information and latch of faults from critical modules on the device
- Collection of self-test results
- Configurable and graded fault control
  - Internal reactions (no internal reaction, IRQ)
  - External reaction (failure is reported to the external/surrounding system via configurable output pins)

1.7.15 System integration unit (SIUL)

The SPC56xP60/54 SIUL controls MCU pad configuration, external interrupts, general purpose I/O (GPIO) pin configuration, and internal peripheral multiplexing.

The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the I/O pins of the MCU.

The SIUL provides the following features:

- Centralized general purpose input output (GPIO) control of input/output pins and analog input-only pads (package dependent)
- All GPIO pins can be independently configured to support pull-up, pull down, or no pull
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- All peripheral pins (except ADC channels) can be alternatively configured as both general purpose input or output pins
- ADC channels support alternative configuration as general purpose inputs
- Direct readback of the pin value is supported on all pins through the SIU
- Configurable digital input filter that can be applied to some general purpose input pins for noise elimination
  - Up to 4 internal functions can be multiplexed onto one pin

1.7.16 Boot and censorship

Different booting modes are available in the SPC56xP60/54:

- From internal flash memory
- Via a serial link

The default booting scheme is the one which uses the internal flash memory (an internal pull-down is used to select this mode). The alternate option allows the user to boot via FlexCAN or LINFlex (using the boot assist module software).

A censorship scheme is provided to protect the contents of the flash memory and offer increased security for the entire device.

A password mechanism is designed to grant the legitimate user access to the non-volatile memory.

Boot assist module (BAM)

The BAM is a block of read-only one-time programmed memory and is identical for all SPC56xP60/54 devices that are based on the e200z0h core. The BAM program is executed every time the device is powered on if the alternate boot mode has been selected by the user.

The BAM provides the following features:

- Serial bootloading via FlexCAN or LINFlex.
- BAM can accept a password via the used serial communication channel to grant the legitimate user access to the non-volatile memory.

1.7.17 Error correction status module (ECSM)

The ECSM on this device features the following:

- Platform configuration and revision
- ECC error reporting for flash memory and SRAM
- ECC error injection for SRAM

The ECSM module is replicated for each processor.

1.7.18 FlexCAN

The FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B. The CAN protocol was designed to be used primarily as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. FlexCAN module contains 32 message buffers.

The FlexCAN module provides the following features:

- Full implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - 0 to 8 bytes data length
  - Programmable bit rate as fast as 1 Mbit/s
- 32 message buffers of 0 to 8 bytes data length
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages
- Programmable loop-back mode supporting self-test operation
- 3 programmable mask registers
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages
- Transmit features
  - Supports configuration of multiple mailboxes to form message queues of scalable depth
  - Arbitration scheme according to message ID or message buffer number
  - Internal arbitration to guarantee no inner or outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - 8 mailboxes configurable as a six-entry receive FIFO
  - 8 programmable acceptance filters for receive FIFO
- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid PLL jitter

1.7.19 Safety port (FlexCAN)

The SPC56xP60/54 MCU has a second CAN controller synthesized to run at high bit rates to be used as a safety port. The CAN module of the safety port provides the following features:

- Identical to the FlexCAN module
- Bit rate as fast as 7.5 Mb at 60 MHz CPU clock using direct connection between CAN modules (no physical transceiver required)
- 32 Message buffers of 0 to 8 bytes data length
- Can be used as a third independent CAN module

1.7.20 FlexRay

The FlexRay module provides the following features:

- Full implementation of FlexRay Protocol Specification 2.1
- 64 configurable message buffers can be handled
- Dual channel or single channel mode of operation, each as fast as 10 Mbit/s data rate
- Message buffers configurable as Tx, Rx or RxFIFO
- Message buffer size configurable
- Message filtering for all message buffers based on FrameID, cycle count and message ID
- Programmable acceptance filters for RxFIFO message buffers

1.7.21 Serial communication interface module (LINFlex)

The LINFlex on the SPC56xP60/54 features the following:

- Supports LIN Master mode (on both modules), LIN Slave mode (on one module) and UART mode
- LIN state machine compliant to LIN1.3, 2.0, and 2.1 Specifications
- Handles LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store Identifier and up to 8 data bytes
  - Supports message length as long as 64 bytes
  - Detection and flagging of LIN errors: Sync field; Delimiter; ID parity; Bit; Framing; Checksum and Time-out errors
  - Classic or extended checksum calculation
  - Configurable Break duration as long as 36-bit times
  - Programmable Baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features: Loop back; Self Test; LIN bus stuck dominant detection
  - Interrupt-driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling
- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-byte receive, 4-byte transmit
  - Configurable word length (8-bit or 9-bit words)
  - Error detection and flagging
  - Parity, Noise and Framing errors
  - Interrupt-driven operation with four interrupt sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud-rate modulus counter and 16-bit fractional
  - 2 receiver wake-up methods

1.7.22 Deserial serial peripheral interface (DSPI)

The deserial serial peripheral interface (DSPI) module provides a synchronous serial interface for communication between the SPC56xP60/54 MCU and external devices.

The DSPI modules provide these features:

- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- Up to 28 chip select lines available
  - 8 each on DSPI_0 and DSPI_1
  - 4 each on DSPI_2, DSPI_3, and DSPI_4
- 8 clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for deglitching
- FIFOs for buffering up to 5 transfers on the transmit and receive side
- Queueing operation possible through use of the eDMA
- General purpose I/O functionality on pins when not used for SPI

1.7.23 eTimer

Two eTimer modules are provided, each with six 16-bit general purpose up/down timer/counter per module. The following features are implemented:

- Individual channel capability
  - Input capture trigger
  - Output compare
  - Double buffer (to capture rising edge and falling edge)
  - Separate prescaler for each counter
  - Selectable clock source
  - 0% to 100% pulse measurement
  - Rotation direction flag (Quad decoder mode)
- Maximum count rate
  - Equals peripheral clock/2 — for external event counting
  - Equals peripheral clock — for internal clock counting
- Cascadeable counters
- Programmable count modulo
- Quadrature decode capabilities
- Counters can share available input pins
- Count once or repeatedly
- Preloadable counters
- Pins available as GPIO when timer functionality not in use

1.7.24 Analog-to-digital converter (ADC)

The ADC module provides the following features:

Analog part:

- 1 on-chip analog-to-digital converter
- 10-bit AD resolution
- 1 sample and hold unit per ADC
- Conversion time, including sampling time, less than 1 µs (at full precision)
- Typical sampling time is 150 ns min. (at full precision)
- Differential non-linearity error (DNL) ±1 LSB
- Integral non-linearity error (INL) ±1.5 LSB
- Total unadjusted error (TUE) <3 LSB
- Single-ended input signal range from 0 to 3.3 V / 5.0 V
- The ADC and its reference can be supplied with a voltage independent from V_{DDIO}
- The ADC supply can be equal or higher than V_{DDIO}
- The ADC supply and the ADC reference are not independent from each other (they are internally bonded to the same pad)
- Sample times of 2 (default), 8, 64, or 128 ADC clock cycles

Digital part:

- 27 input channels (26 + 1 internally connected)
- 4 analog watchdogs to compare ADC results against predefined levels (low, high, range) before results are stored
- 2 modes of operation: Normal mode or CTU control mode
- Normal mode features
  - Register-based interface with the CPU: control reg., status reg., 1 result register per channel
  - ADC state machine managing 3 request flows: regular command, hardware injected command, software injected command
  - Selectable priority between software and hardware injected commands
  - DMA compatible interface
- CTU control mode features
  - Triggered mode only
  - 4 independent result queues (2 × 16 entries, 2 × 4 entries)
  - Result alignment circuitry (left justified; right justified)
  - 32-bit read mode allows to have channel ID on one of the 16-bit part
  - DMA compatible interfaces

1.7.25 Cross triggering unit (CTU)

The Cross Triggering Unit (CTU) allows automatic generation of ADC conversion requests on user selected conditions with minimized CPU load for dynamic configuration.

It implements the following features:

- Double buffered trigger generation unit with up to eight independent triggers generated from external triggers
- Trigger generation unit configurable in sequential mode or in triggered mode
- Each Trigger can be appropriately delayed to compensate the delay of external low pass filter
- Double buffered global trigger unit allowing eTimer synchronization and/or ADC command generation
- Double buffered ADC command list pointers to minimize ADC-trigger unit update
- Double buffered ADC conversion command list with up to 24 ADC commands
- Each trigger has the capability to generate consecutive commands
- ADC conversion command allows to control ADC channel from each ADC, single or synchronous sampling, independent result queue selection

1.7.26 Cyclic redundancy check (CRC)

- 3 contexts for the concurrent CRC computation
- Separate CRC engine for each context
- Zero-wait states during the CRC computation (pipeline scheme)
- 3 hard-wired polynomials (CRC-8 VDA CAN, CRC-32 ethernet and CRC-16-CCITT)
- Support for byte/half-word/word width of the input data stream
- Support for expected and actual CRC comparison

1.7.27 Nexus development interface (NDI)

The NDI block provides real-time development support capabilities for the SPC56xP60/54 Power Architecture based MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility. The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface for this device. The NDI block interfaces to the host processor and internal buses to provide development support as per the IEEE-ISTO 5001-2003 Class 2+ standard. The development support provided includes access to the MCU's internal memory map and access to the processor's internal registers during run time.

The Nexus Interface provides the following features:

- Configured via the IEEE 1149.1
- All Nexus port pins operate at V_{DDIO} (no dedicated power supply)
- Nexus 2+ features supported
  - Static debug
  - Watchpoint messaging
  - Ownership trace messaging
  - Program trace messaging
  - Real time read/write of any internally memory mapped resources through JTAG pins
  - Overrun control, which selects whether to stall before Nexus overruns or keep executing and allow overwrite of information
  - Watchpoint triggering, watchpoint triggers program tracing
  - DDR
- Auxiliary Output Port
  - 4 MDO (Message Data Out) pins
  - MCKO (Message Clock Out) pin
  - 2 $\overline{\text{MSEO}}$ (Message Start/End Out) pins
  - $\overline{\text{EVTO}}$ (Event Out) pin
- Auxiliary Input Port
  - $\overline{\text{EVTI}}$ (Event In) pin

1.7.28 IEEE 1149.1 (JTAG) controller

The JTAG controller (JTAGC) block provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. All data input to and output from the JTAGC block is communicated in serial format. The JTAGC block is compliant with the IEEE standard.

The JTAG controller provides the following features:

- IEEE Test Access Port (TAP) interface with four pins (TDI, TMS, TCK, TDO)
- Selectable modes of operation include JTAGC/debug or normal system operation.
- A 5-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS, IDCODE, EXTEST, SAMPLE, SAMPLE/PRELOAD
- A 5-bit instruction register that supports the additional following public instructions:
  - ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_CORE0, ACCESS_AUX_TAP_CORE1, ACCESS_AUX_TAP_NASPS_0, ACCESS_AUX_TAP_NASPS_1
- Three test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry.

1.7.29 On-chip voltage regulator (VREG)

The on-chip voltage regulator module provides the following features:

- Uses external NPN transistor
- Regulates external 3.3 V to 5.0 V down to 1.2 V for the core logic
- Low voltage detection on the internal 1.2 V and I/O voltage 3.3 V

1.8 Developer environment

The SPC56xP60/54 MCU tools and third-party developers offer a widespread, established network of tool and software vendors. The device also features a high-performance Nexus debug interface.

The following development support will be available:

- Automotive Evaluation Boards (EVB) featuring CAN, LIN interfaces, and more
- Compilers
- Debuggers
- JTAG and Nexus interfaces
- Autocode generation tools
- Initialization tools

The following software support will be available:

- AUTOSAR[a] OS solutions from multiple third parties
- LIN drivers
- AUTOSAR BSW (Basic Software) Package and Configuration Tool from multiple third parties
- Core and peripheral self test[b]

a. Automotive Open System Architecture

b. Available in third quarter of 2010

2 Memory Map

Table 4 shows the memory map for the SPC56xP60/54.

All addresses on the SPC56xP60/54, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each region or module name.

Table 4. Memory map

Start address	End address	Size (KB)	Region name
On-chip memory			
0x0000_0000	0x0007_FFFF	512	Code Flash Array 0
0x0008_0000	0x000F_FFFF	512	Code Flash Array 1
0x0010_0000	0x001F_FFFF	1024	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash Array 0 Shadow Sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash Array 0 Test Sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_FFFF	64	Data Flash Array 0
0x0081_0000	0x00C0_1FFF	4040	Reserved
0x00C0_2000	0x00C0_3FFF	8	Data Flash Array 0 Test Sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Flash Emulation Mapping
0x2000_0000	0x3FFF_FFFF	524288	Reserved
0x4000_0000	0x4000_BFFF	48	SRAM(1)
0x4000_C000	0x4FFF_FFFF	262096	Reserved
0x5000_0000	0x5000_7FFF	32	SRAM(1)
0x5000_8000	0x7FFF_FFFF	786368	Reserved
On-chip Peripherals(2)			
0x8000_0000	0x8FEFFFFF	261120	Reserved
0x8FF0_0000	0x8FF0_3FFF	16	Peripheral Bridge (AIPS_1)
0x8FF0_4000	0x8FF0_FFFF	48	Reserved
0x8FF1_0000	0x8FF1_3FFF	16	Memory Protection Unit (MPU_1)
0x8FF1_4000	0x8FF2_3FFF	64	Reserved
0x8FF2_4000	0x8FF2_7FFF	16	Semaphores (SEMA4_1)
0x8FF2_8000	0x8FF3_7FFF	64	Reserved
0x8FF3_8000	0x8FF3_BFFF	16	Software Watchdog (SWT_1)

Table 4. Memory map (continued)

Start address	End address	Size (KB)	Region name
0x8FF3_C000	0x8FF3_FFFF	16	System Timer Module (STM_1)
0x8FF4_0000	0x8FF4_3FFF	16	Error Correction Status Module (ECSM_1)
0x8FF4_4000	0x8FF4_7FFF	16	Reserved
0x8FF4_8000	0x8FF4_BFFF	16	Interrupt Controller (INTC_1)
0x8FF4_C000	0x8FF9_FFFF	336	Reserved
0x8FFA_0000	0x8FFA_3FFF	16	Deserial Serial Peripheral Interface (DSPI_4)
0x8FFA_4000	0x8FFC_3FFF	128	Reserved
0x8FFC_4000	0x8FFC_7FFF	16	FlexCAN 1 (CAN_1)
0x8FFC_8000	0x9FE6_BFFF	260752	Reserved
0x9FE6_C000	0x9FE6_FFFF	16	Fault Collection and Control Unit (FCCU)
0x9FE7_0000	0x9FE7_3FFF	16	Cyclic Redundancy Check Unit (CRC_1)
0x9FE7_4000	0xC3F8_7FFF	590928	Reserved
0xC3F8_8000	0xC3F8_BFFF	16	Code Flash 0 Configuration (CFLASH0)
0xC3F8_C000	0xC3F8_FFFF	16	Data Flash 0 Configuration (DFLASH0)
0xC3F9_0000	0xC3F9_3FFF	16	System Integration Unit Lite (SIUL)
0xC3F9_4000	0xC3F9_7FFF	16	WakeUp Unit (WKPU)
0xC3F9_8000	0xC3FD_7FFF	256	Reserved
0xC3FD_8000	0xC3FD_BFFF	16	System Status and Configuration Module (SSCM)
0xC3FD_C000	0xC3FD_FFFF	16	Mode Entry Module (MC_ME)
0xC3FE_0000	0xC3FE_3FFF	16	Clock Generation Module (MC_CGM)
0xC3FE_4000	0xC3FE_7FFF	16	Reset Generation Module (MC_RGM)
0xC3FE_8000	0xC3FE_BFFF	16	Power Control Unit (MC_PCU)(3)
0xC3FE_C000	0xC3FE_FFFF	16	Reserved
0xC3FF_0000	0xC3FF_3FFF	16	Periodic Interrupt Timer (PIT)
0xC3FF_4000	0xC3FF_FFFF	48	Reserved
0xFFE0_0000	0xFFE0_3FFF	16	Analog-to-digital Converter 0 (ADC_0)
0xFFE0_4000	0xFFE0_BFFF	32	Reserved
0xFFE0_C000	0xFFE0_FFFF	16	CTU_0
0xFFE1_0000	0xFFE1_7FFF	32	Reserved
0xFFE1_8000	0xFFE1_BFFF	16	eTimer_0
0xFFE1_C000	0xFFE1_FFFF	16	eTimer_1
0xFFE2_0000	0xFFE3_FFFF	128	Reserved
0xFFE4_0000	0xFFE4_3FFF	16	LINFlex_0
0xFFE4_4000	0xFFE4_7FFF	16	LINFlex_1

Table 4. Memory map (continued)

Start address	End address	Size (KB)	Region name
0xFFE5_0000	0xFFE6_7FFF	128	Reserved
0xFFE6_8000	0xFFE6_BFFF	16	Cyclic Redundancy Check (CRC_0)
0xFFE6_C000	0xFFE7_FFFF	80	Reserved
0xFFE8_0000	0xFFEF_FFFF	512	Mirrored (range 0xC3F8_0000 – 0xC3FF_FFFF)
0xFFF0_0000	0xFFF0_3FFF	16	Peripheral Bridge (AIPS_0)
0xFFF0_4000	0xFFF0_7FFF	16	Crossbar switch (XBAR_0)
0xFFF0_8000	0xFFF0_FFFF	32	Reserved
0xFFF1_0000	0xFFF1_3FFF	16	Memory Protection Unit (MPU_0)
0xFFF1_4000	0xFFF2_3FFF	64	Reserved
0xFFF2_4000	0xFFF2_7FFF	16	Semaphores (SEMA4_0)
0xFFF2_8000	0xFFF3_7FFF	64	Reserved
0xFFF3_8000	0xFFF3_BFFF	16	Software Watchdog Timer (SWT_0)
0xFFF3_C000	0xFFF3_FFFF	16	System Timer Module (STM_0)
0xFFF4_0000	0xFFF4_3FFF	16	Error Correction Status Module (ECSM)
0xFFF4_4000	0xFFF4_7FFF	16	Enhanced Direct Memory Access Controller (eDMA)
0xFFF4_8000	0xFFF4_BFFF	16	Interrupt Controller (INTC_0)
0xFFF4_C000	0xFFF8_FFFF	272	Reserved
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_0
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_1
0xFFF9_8000	0xFFF9_BFFF	16	DSPI_2
0xFFF9_C000	0xFFF9_FFFF	16	DSPI_3
0xFFFA_0000	0xFFFB_FFFF	128	Reserved
0xFFFC_0000	0xFFFC_3FFF	16	FlexCAN 0 (CAN_0)
0xFFFC_4000	0xFFFD_BFFF	96	Reserved
0xFFFD_C000	0xFFFD_FFFF	16	DMA Channel Mux (DMA_MUX)
0xFFFE_0000	0xFFFE_3FFF	16	FlexRay Controller (FlexRay)
0xFFFE_4000	0xFFFE_7FFF	16	Reserved
0xFFFE_8000	0xFFFE_BFFF	16	Safety Port (FlexCAN)
0xFFFE_C000	0xFFFF_BFFF	64	Reserved
0xFFFF_C000	0xFFFF_FFFF	16	Boot Assist Module (BAM)

1. For SRAM details please see *Chapter 16: Internal Static RAM (SRAM)*
2. In order to produce Data Abort exception by accessing to the Reserved blocks inside memory 0x8000_0000 - 0xFFFF_FFFF, the SSCM.ERROR[PAE] bit has to be set to one. See *Section : Error Configuration (ERROR) register*.
3. This address space contains also VREG registers. See *Chapter 36: Voltage Regulators and Power Supplies*

3 Signal Description

This chapter describes the signals of the SPC56xP60/54. It includes a table of signal properties and detailed descriptions of signals.

3.1 176-pin LQFP pinout

Figure 3 shows the pinout of the 176-pin LQFP.

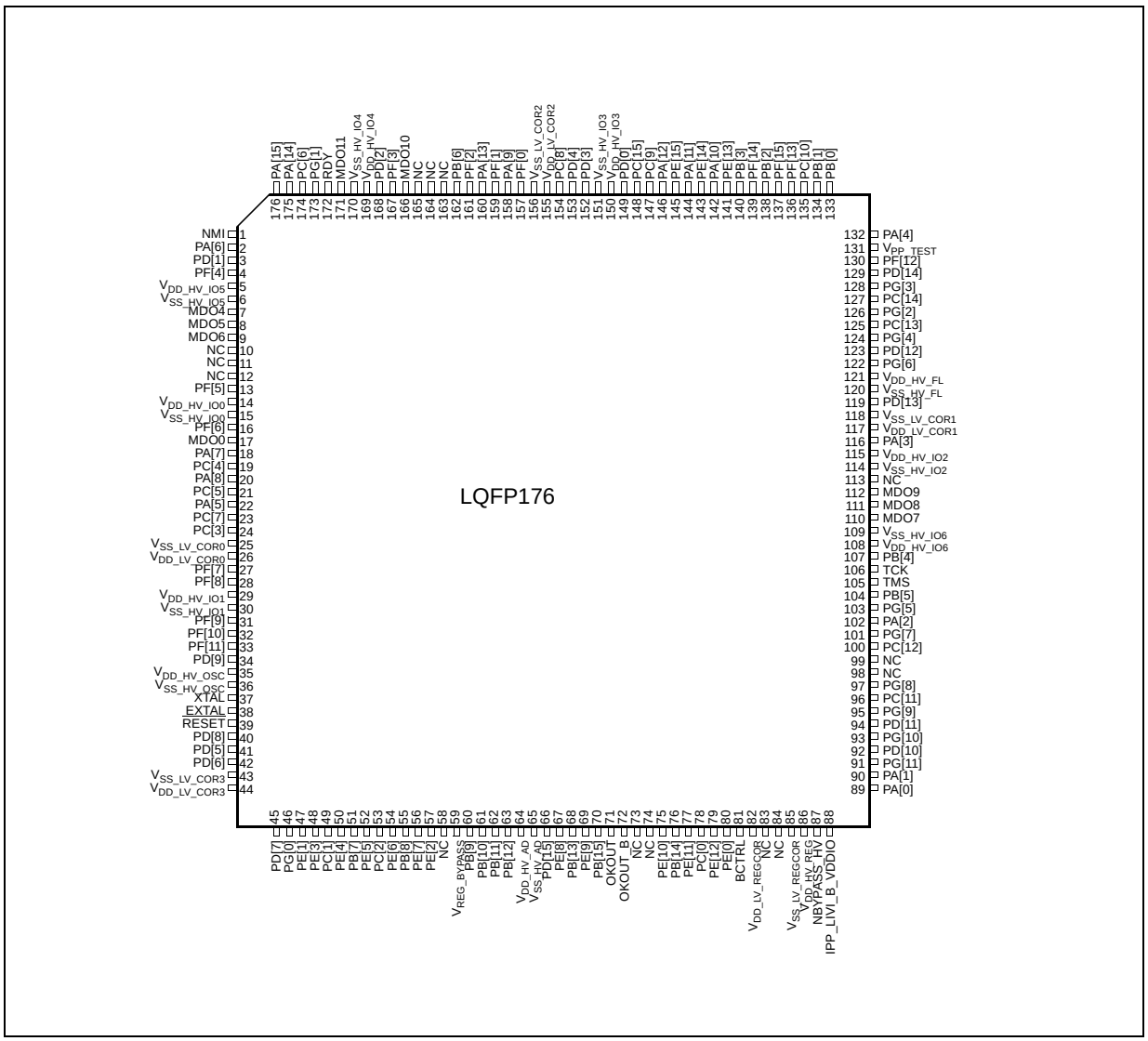


Figure 3. LQFP176 pinout (top view)(c)

c. Software development package only. Not available for production.

3.2 144-pin LQFP pinout

Figure 4 shows the pinout of the 144-pin LQFP.

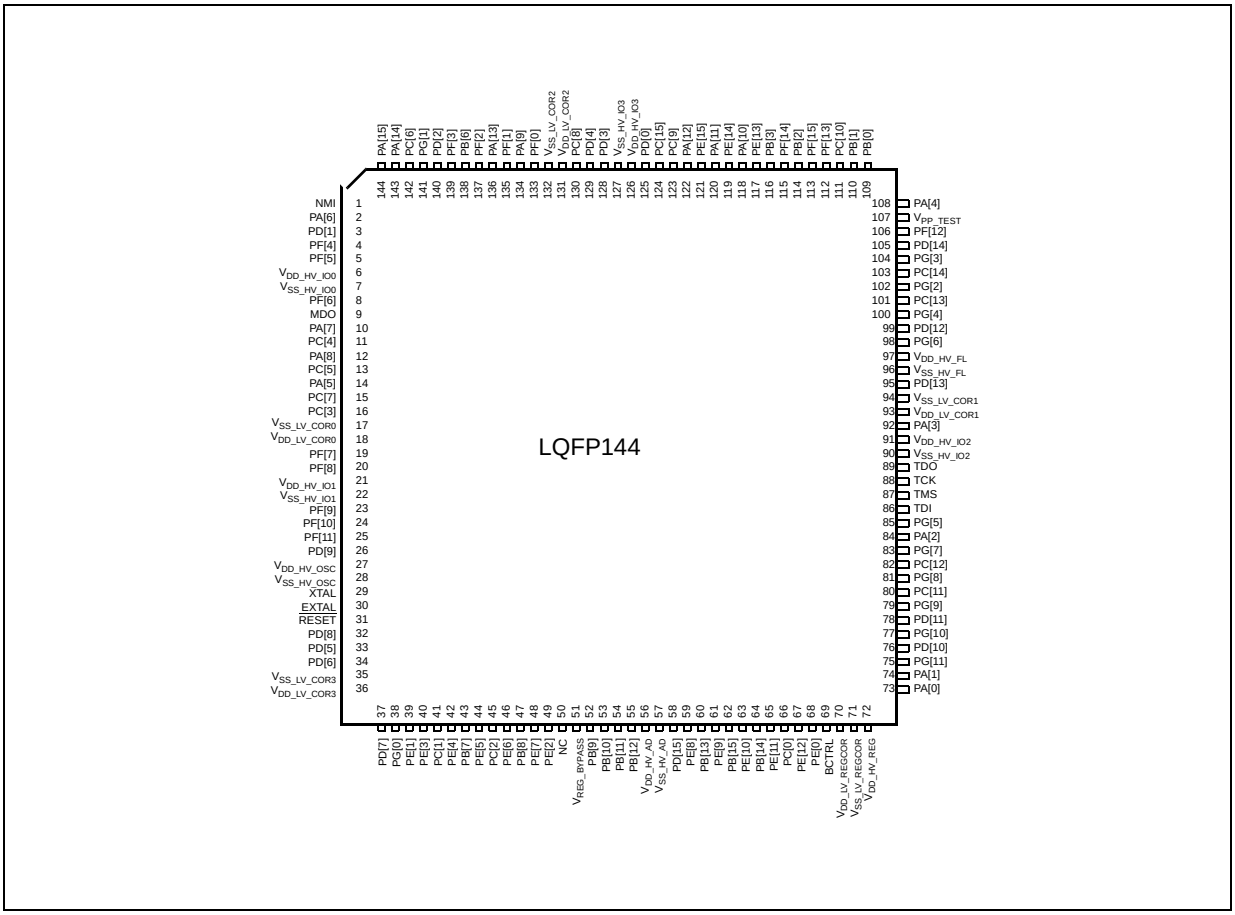


Figure 4. LQFP144 pinout (top view)[d]

d. Availability of port pin alternate functions depends on product selection

3.3 100-pin LQFP pinout

Figure 5 shows the pinout of the 100-pin LQFP.

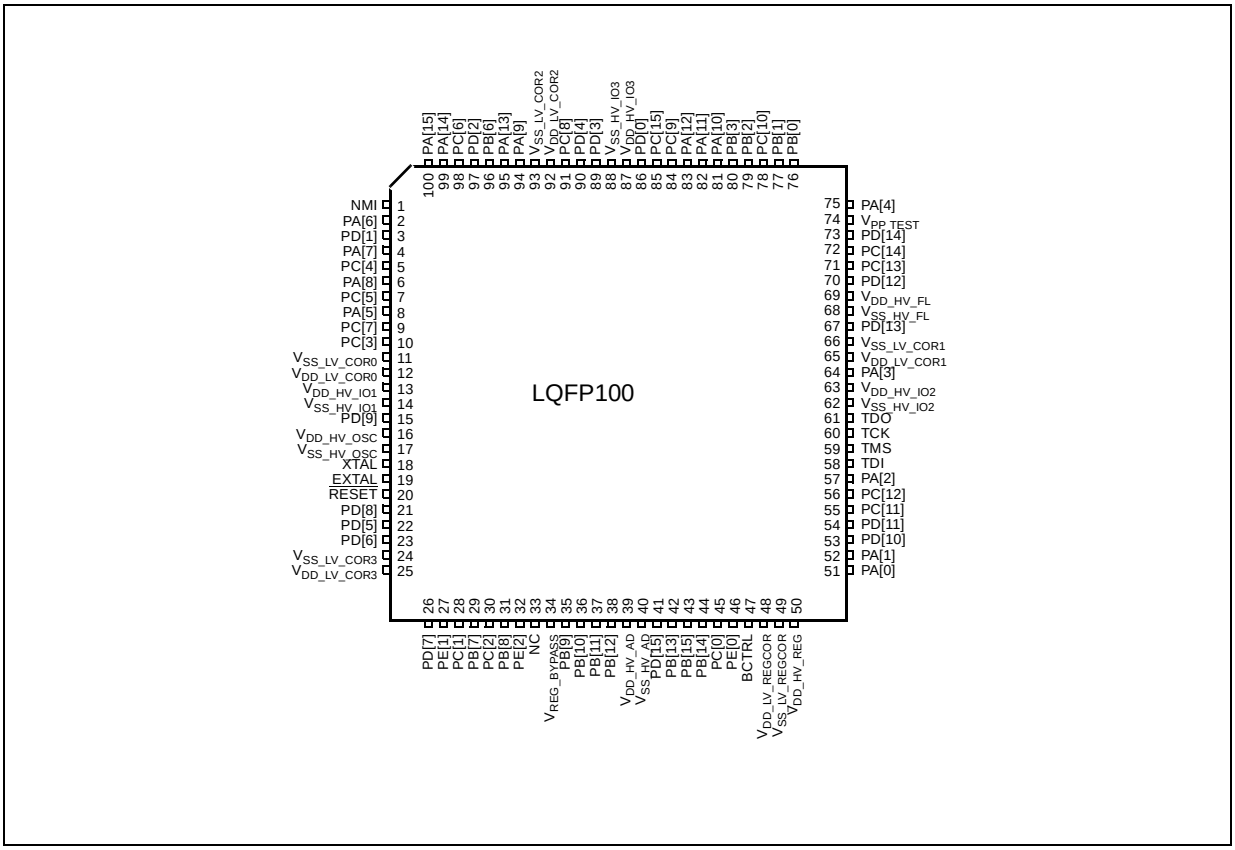


Figure 5. LQFP100 pinout (top view)(e)

e. Availability of port pin alternate functions depends on product selection

3.4 Pin descriptions

The following sections provide signal descriptions and related information about the functionality and configuration of the SPC56xP60/54 devices.

3.4.1 Power supply and reference voltage pins

Table 5 lists the power supply and reference voltage for the SPC56xP60/54 devices.

Table 5. Supply pins

Supply		Pin		
Symbol	**Description**	**LQFP 100**	**LQFP 144**	**LQFP 176(1)**
VREG control and power supply pins				
BCTRL	Voltage regulator external NPN Ballast base control pin	47	69	81
$V_{DD\_HV\_REG}$ (3.3 V or 5.0 V)	Voltage regulator supply voltage	50	72	86
$V_{DD\_LV\_REGCOR}$	1.2 V decoupling(2) pins for core logic supply and voltage regulator feedback. Decoupling capacitor must be connected between this pins and $V_{SS\_LV\_REGCOR}$.	48	70	82
$V_{SS\_LV\_REGCOR}$	1.2 V decoupling(2) pins for core logic GND and voltage regulator feedback. Decoupling capacitor must be connected between this pins and $V_{DD\_LV\_REGCOR}$.	49	71	85
ADC0 reference and supply voltage				
$V_{DD\_HV\_AD}$	ADC supply and high reference voltage	39	56	64
$V_{SS\_HV\_AD}$	ADC ground and low reference voltage	40	57	65
Power supply pins (3.3 V or 5.0 V)				
$V_{DD\_HV\_IO0}$	Input/Output supply voltage	—	6	14
$V_{SS\_HV\_IO0}$	Input/Output ground	—	7	15
$V_{DD\_HV\_IO1}$	Input/Output supply voltage	13	21	29
$V_{SS\_HV\_IO1}$	Input/Output ground	14	22	30
$V_{DD\_HV\_IO2}$	Input/Output supply voltage	63	91	115
$V_{SS\_HV\_IO2}$	Input/Output ground	62	90	114
$V_{DD\_HV\_IO3}$	Input/Output supply voltage	87	126	150
$V_{SS\_HV\_IO3}$	Input/Output ground	88	127	151
$V_{DD\_HV\_IO4}$	Input/Output supply voltage	—	—	169
$V_{SS\_HV\_IO4}$	Input/Output ground	—	—	170
$V_{DD\_HV\_IO5}$	Input/Output supply voltage	—	—	5
$V_{SS\_HV\_IO5}$	Input/Output ground	—	—	6
$V_{DD\_HV\_IO6}$	Input/Output supply voltage	—	—	108
$V_{SS\_HV\_IO6}$	Input/Output ground	—	—	109

Table 5. Supply pins (continued)

Supply		Pin		
Symbol	**Description**	**LQFP 100**	**LQFP 144**	**LQFP 176(1)**
$V_{DD\_HV\_FL}$	Code and data flash supply voltage	69	97	121
$V_{SS\_HV\_FL}$	Code and data flash supply ground	68	96	120
$V_{DD\_HV\_OSC}$	Crystal oscillator amplifier supply voltage	16	27	35
$V_{SS\_HV\_OSC}$	Crystal oscillator amplifier ground	17	28	36
Power supply pins (1.2 V)				
$V_{DD\_LV\_COR0}$	1.2 V Decoupling pins for core logic supply. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR0}$ pin.	12	18	26
$V_{SS\_LV\_COR0}$	1.2 V Decoupling pins for core logic GND. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR0}$ pin.	11	17	25
$V_{DD\_LV\_COR1}$	1.2 V Decoupling pins for core logic supply. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR1}$ pin.	65	93	117
$V_{SS\_LV\_COR1}$	1.2 V Decoupling pins for core logic GND. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR1}$ pin.	66	94	118
$V_{DD\_LV\_COR2}$	1.2 V Decoupling pins for core logic supply. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR2}$ pin.	92	131	155
$V_{SS\_LV\_COR2}$	1.2 V Decoupling pins for core logic GND. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR\ 2}$ pin.	93	132	156
$V_{DD\_LV\_COR3}$	1.2 V Decoupling pins for core logic supply. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR3}$ pin.	25	36	44
$V_{SS\_LV\_COR3}$	1.2 V Decoupling pins for core logic GND. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR\ 3}$ pin.	24	35	43

1. LQFP176 available only as development package.
2. See datasheet Voltage Regulator Electrical Characteristics section for more details.

3.4.2 System pins

Table 6 and *Table 7* contain information on pin functions for the SPC56xP60/54 devices. The pins listed in *Table 6* are single-function pins. The pins shown in *Table 7* are multi-function pins, programmable via their respective Pad Configuration Register (PCR) values.

Table 6. System pins

Symbol	Description	Direction	Pad Speed(1)		Pin		
			SRC=0	SRC=1	LQFP 100	LQFP 144	LQFP 176(2)
Dedicated pins							
MDO0	Nexus Message Data Output—line 0	Output Only	Fast		—	9	17
MDO4	Nexus Message Data Output—line 4	Output Only	Fast		—	—	7
MDO5	Nexus Message Data Output—line 5	Output Only	Fast		—	—	8
MDO6	Nexus Message Data Output—line 6	Output Only	Fast		—	—	9
MDO7	Nexus Message Data Output—line 7	Output Only	Fast		—	—	110
MDO8	Nexus Message Data Output—line 8	Output Only	Fast		—	—	111
MDO9	Nexus Message Data Output—line 9	Output Only	Fast		—	—	112
MDO10	Nexus Message Data Output—line 10	Output Only	Fast		—	—	166
MDO11	Nexus Message Data Output—line 11	Output Only	Fast		—	—	171
RDY	Nexus ready output	Output Only	—	—	—	—	172
NMI	Non-Maskable Interrupt	Input Only	—	—	1	1	1
XTAL	Analog output of the oscillator amplifier circuit. Needs to be grounded if oscillator is used in bypass mode.	—	—	—	18	29	37
EXTAL	Analog input of the oscillator amplifier circuit, when the oscillator is not in bypass mode. Analog input for the clock generator when the oscillator is in bypass mode.	—	—	—	19	30	38
TMS(3)	JTAG state machine control	Input Only	—	—	59	87	105
TCK(3)	JTAG clock	Input Only	—	—	60	88	106
TDI(3)	JTAG data input	Input Only	—	—	58	86	104
TDO(3)	JTAG data output	Output Only	—	—	61	89	107
Reset pin							
RESET(4)	Bidirectional reset with Schmitt trigger characteristics and noise filter	Bidirectional	Medium	—	20	31	39
Test pin							
$V_{PP\ TEST}$	Pin for testing purpose only. To be tied to ground in normal operating mode.	—	—	—	74	107	131
V_{REG_BYPASS}	Pin for testing purpose only. To be tied to ground in normal operating mode.	—	—	—	34	51	59

1. SRC values refer to the value assigned to the Slew Rate Control bits of the pad configuration register.
2. LQFP176 available only as development package.
3. In this pin there is an internal pull, refer to JTAGC chapter in the device reference manual for pull direction.

4. Its configuration can be set up by the PCR[108] register inside the SIU module. See *Chapter 11: System Integration Unit Lite (SIUL)*

3.4.3 Pin muxing

Table 7 defines the pin list and muxing for the SPC56xP60/54 devices relative to Enhanced Full-featured version.

Each row of *Table 7* shows all the possible ways of configuring each pin, via “alternate functions”. The default function assigned to each pin after reset is the ALT0 function.

Pins marked as external interrupt capable can also be used to resume from STOP and HALT mode.

SPC56xP60/54 devices provide four main I/O pad types depending on the associated functions:

- *Slow pads* are the most common, providing a compromise between transition time and low electromagnetic emission.
- *Medium pads* provide fast enough transition for serial communication channels with controlled current to reduce electromagnetic emission.
- *Fast pads* provide maximum speed. They are used for improved NEXUS debugging capability.
- *Symmetric pads* are designed to meet FlexRay requirements.

Medium and Fast pads can use slow configuration to reduce electromagnetic emission, at the cost of reducing AC performance.

Table 7. Pin muxing(1)

Port pin	PCR register	Alternate function(2), (3)	Functions	Peripheral (4)	I/O direction (5)	Pad speed(6)		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176(7)
Port A										
A[0]	PCR[0]	ALT0 ALT1 ALT2 ALT3 —	GPIO[0] ETC[0] SCK_2 F[0] EIRQ[0]	SIUL eTimer_0 DSPI_2 FCCU SIUL	I/O I/O I/O O I	Slow	Medium	51	73	89
A[1]	PCR[1]	ALT0 ALT1 ALT2 ALT3 —	GPIO[1] ETC[1] SOUT_2 F[1] EIRQ[1]	SIUL eTimer_0 DSPI_2 FCCU SIUL	I/O I/O O O I	Slow	Medium	52	74	90

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
A[2][8]	PCR[2]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[2] ETC[2] CS3 — SIN_2 ABS[0] EIRQ[2]	SIUL eTimer_0 DSPI_4 — DSPI_2 MC_RGM SIUL	I/O I/O O — I I I	Slow	Medium	57	84	102
A[3][8]	PCR[3]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[3] ETC[3] CS0_2 — ABS[1] EIRQ[3]	SIUL eTimer_0 DSPI_2 — MC_RGM SIUL	I/O I/O I/O — I I	Slow	Medium	64	92	116
A[4][8]	PCR[4]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[4] ETC[0] CS1_2 ETC[4] FAB EIRQ[4]	SIUL eTimer_1 DSPI_2 eTimer_0 MC_RGM SIUL	I/O I/O O I/O I I	Slow	Medium	75	108	132
A[5]	PCR[5]	ALT0 ALT1 ALT2 ALT3 —	GPIO[5] CS0_1 ETC[5] CS7_0 EIRQ[5]	SIUL DSPI_1 eTimer_1 DSPI_0 SIUL	I/O I/O I/O O I	Slow	Medium	8	14	22
A[6]	PCR[6]	ALT0 ALT1 ALT2 ALT3 —	GPIO[6] SCK_1 CS2_4 — EIRQ[6]	SIUL DSPI_1 DSPI_4 — SIUL	I/O I/O I/O — I	Slow	Medium	2	2	2
A[7]	PCR[7]	ALT0 ALT1 ALT2 ALT3 —	GPIO[7] SOUT_1 CS1_4 — EIRQ[7]	SIUL DSPI_1 DSPI_4 — SIUL	I/O O I/O — I	Slow	Medium	4	10	18
A[8]	PCR[8]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[8] — CS0_4 — SIN_1 EIRQ[8]	SIUL — DSPI_4 — DSPI_1 SIUL	I/O — I/O — I I	Slow	Medium	6	12	20

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
A[9]	PCR[9]	ALT0 ALT1 ALT2 ALT3 —	GPIO[9] CS1_2 — — SIN_4	SIUL DSPI_2 — — DSPI_4	I/O O — — I	Slow	Medium	94	134	158
A[10]	PCR[10]	ALT0 ALT1 ALT2 ALT3 —	GPIO[10] CS0_2 — — EIRQ[9]	SIUL DSPI_2 — — SIUL	I/O I/O — — I	Slow	Medium	81	118	142
A[11]	PCR[11]	ALT0 ALT1 ALT2 ALT3 —	GPIO[11] SCK_2 — — EIRQ[10]	SIUL DSPI_2 — — SIUL	I/O I/O — — I	Slow	Medium	82	120	144
A[12]	PCR[12]	ALT0 ALT1 ALT2 ALT3 —	GPIO[12] SOUT_2 — — EIRQ[11]	SIUL DSPI_2 — — SIUL	I/O O — — I	Slow	Medium	83	122	146
A[13]	PCR[13]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[13] CS4_1 — — SIN_2 EIRQ[12]	SIUL DSPI_1 — — DSPI_2 SIUL	I/O — — — I I	Slow	Medium	95	136	160
A[14]	PCR[14]	ALT0 ALT1 ALT2 ALT3 —	GPIO[14] TXD ETC[4] CS5_1 EIRQ[13]	SIUL Safety Port eTimer_1 DSPI_1 SIUL	I/O O I/O O I	Slow	Medium	99	143	175
A[15]	PCR[15]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[15] CS6_1 ETC[5] — RXD EIRQ[14]	SIUL DSPI_1 eTimer_1 — Safety Port SIUL	I/O O I/O — I I	Slow	Medium	100	144	176

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2], [3]	Functions	Peripheral [4]	I/O direction [5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
Port B										
B[0]	PCR[16]	ALT0 ALT1 ALT2 ALT3 —	GPIO[16] TXD ETC[2] DEBUG[0] EIRQ[15]	SIUL FlexCAN_0 eTimer_1 SSCM SIUL	I/O O I/O — I	Slow	Medium	76	109	133
B[1]	PCR[17]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[17] CS7_1 ETC[3] DEBUG[1] RXD EIRQ[16]	SIUL DSPI_1 eTimer_1 SSCM FlexCAN_0 SIUL	I/O O I/O — I I	Slow	Medium	77	110	134
B[2]	PCR[18]	ALT0 ALT1 ALT2 ALT3 —	GPIO[18] TXD SOUT_4 DEBUG[2] EIRQ[17]	SIUL LINFlex_0 DSPI_4 SSCM SIUL	I/O O I/O — I	Slow	Medium	79	114	138
B[3]	PCR[19]	ALT0 ALT1 ALT2 ALT3 —	GPIO[19] — SCK_4 DEBUG[3] RXD	SIUL — DSPI_4 SSCM LINFlex_0	I/O — I/O — I	Slow	Medium	80	116	140
B[6]	PCR[22]	ALT0 ALT1 ALT2 ALT3 —	GPIO[22] clk_out CS2_2 clk_out_div256 EIRQ[18]	SIUL MC_CGL DSPI_2 MC_CGL SIUL	I/O O O O I	Slow	Medium	96	138	162
B[7]	PCR[23]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[23] — — — AN[0] RXD	SIUL — — — ADC_0 LINFlex_0	Input Only	—	—	29	43	51
B[8]	PCR[24]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[24] — — — AN[1] ETC[5]	SIUL — — — ADC_0 eTimer_0	Input Only	—	—	31	47	55

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
B[9]	PCR[25]	ALT0 ALT1 ALT2 ALT3 —	GPIO[25] — — — AN[11]	SIUL — — — ADC_0	Input Only	—	—	35	52	60
B[10]	PCR[26]	ALT0 ALT1 ALT2 ALT3 —	GPIO[26] — — — AN[12]	SIUL — — — ADC_0	Input Only	—	—	36	53	61
B[11]	PCR[27]	ALT0 ALT1 ALT2 ALT3 —	GPIO[27] — — — AN[13]	SIUL — — — ADC_0	Input Only	—	—	37	54	62
B[12]	PCR[28]	ALT0 ALT1 ALT2 ALT3 —	GPIO[28] — — — AN[14]	SIUL — — — ADC_0	Input Only	—	—	38	55	63
B[13]	PCR[29]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[29] — — — AN[16] RXD	SIUL — — — ADC_0 LINFlex_1	Input Only	—	—	42	60	68
B[14]	PCR[30]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[30] — — — AN[17] ETC[4] EIRQ[19]	SIUL — — — ADC_0 eTimer_0 SIUL	Input Only	—	—	44	64	76
B[15]	PCR[31]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[31] — — — AN[18] EIRQ[20]	SIUL — — — ADC_0 SIUL	Input Only	—	—	43	62	70

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
Port C										
C[0]	PCR[32]	ALT0 ALT1 ALT2 ALT3 —	GPIO[32] — — — AN[19]	SIUL — — — ADC_0	Input Only	—	—	45	66	78
C[1]	PCR[33]	ALT0 ALT1 ALT2 ALT3 —	GPIO[33] — — — AN[2]	SIUL — — — ADC_0	Input Only	—	—	28	41	49
C[2]	PCR[34]	ALT0 ALT1 ALT2 ALT3 —	GPIO[34] — — — AN[3]	SIUL — — — ADC_0	Input Only	—	—	30	45	53
C[3]	PCR[35]	ALT0 ALT1 ALT2 ALT3 —	GPIO[35] CS1_0 ETC[4] TXD EIRQ[21]	SIUL DSPI_0 eTimer_1 LINFlex_1 SIUL	I/O O I/O O I	Slow	Medium	10	16	24
C[4]	PCR[36]	ALT0 ALT1 ALT2 ALT3 —	GPIO[36] CS0_0 — DEBUG[4] EIRQ[22]	SIUL DSPI_0 — SSCM SIUL	I/O I/O — — I	Slow	Medium	5	11	19
C[5]	PCR[37]	ALT0 ALT1 ALT2 ALT3 —	GPIO[37] SCK_0 SCK_4 DEBUG[5] EIRQ[23]	SIUL DSPI_0 DSPI_4 SSCM SIUL	I/O I/O I/O — I	Slow	Medium	7	13	21
C[6]	PCR[38]	ALT0 ALT1 ALT2 ALT3 —	GPIO[38] SOUT_0 — DEBUG[6] EIRQ[24]	SIUL DSPI_0 — SSCM SIUL	I/O O — — I	Slow	Medium	98	142	174

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
C[7]	PCR[39]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[39] — — DEBUG[7] SIN_0 SIN_4	SIUL — — SSCM DSPI_0 DSPI_4	I/O — — — I I	Slow	Medium	9	15	23
C[8]	PCR[40]	ALT0 ALT1 ALT2 ALT3	GPIO[40] CS1_1 CS1_4 CS6_0	SIUL DSPI_1 DSPI_4 DSPI_0	I/O O O O	Slow	Medium	91	130	154
C[9]	PCR[41]	ALT0 ALT1 ALT2 ALT3	GPIO[41] CS3_2 CS0_4 —	SIUL DSPI_2 DSPI_4 —	I/O O I/O —	Slow	Medium	84	123	147
C[10]	PCR[42]	ALT0 ALT1 ALT2 ALT3	GPIO[42] CS2_2 CS2_4 —	SIUL DSPI_2 DSPI_4 —	I/O O O —	Slow	Medium	78	111	135
C[11]	PCR[43]	ALT0 ALT1 ALT2 ALT3	GPIO[43] ETC[4] CS2_2 CS0_3	SIUL eTimer_0 DSPI_2 DSPI_3	I/O I/O O I/O	Slow	Medium	55	80	96
C[12]	PCR[44]	ALT0 ALT1 ALT2 ALT3	GPIO[44] ETC[5] CS3_2 CS1_3	SIUL eTimer_0 DSPI_2 DSPI_3	I/O I/O O O	Slow	Medium	56	82	100
C[13]	PCR[45]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[45] ETC[1] — — EXT_IN RXD	SIUL eTimer_1 — — CTU_0 FlexCAN_1	I/O I/O — — I I	Slow	Medium	71	101	125
C[14]	PCR[46]	ALT0 ALT1 ALT2 ALT3	GPIO[46] ETC[2] EXT_TGR TXD	SIUL eTimer_1 CTU_0 FlexCAN_1	I/O I/O O O	Slow	Medium	72	103	127

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
C[15]	PCR[47]	ALT0 ALT1 ALT2 ALT3 —	GPIO[47] CA_TR_EN ETC[0] — EXT_IN	SIUL FlexRay_0 eTimer_1 — CTU_0	I/O O I/O — I	Slow	Symmetric	85	124	148
Port D										
D[0]	PCR[48]	ALT0 ALT1 ALT2 ALT3	GPIO[48] CA_TX ETC[1] —	SIUL FlexRay_0 eTimer_1 —	I/O O I/O —	Slow	Symmetric	86	125	149
D[1]	PCR[49]	ALT0 ALT1 ALT2 ALT3 —	GPIO[49] CS4_1 ETC[2] EXT_TRG CA_RX	SIUL DSPI_1 eTimer_1 CTU_0 FlexRay_0	I/O O I/O O I	Slow	Medium	3	3	3
D[2]	PCR[50]	ALT0 ALT1 ALT2 ALT3 —	GPIO[50] CS5_1 ETC[3] — CB_RX	SIUL DSPI_1 eTimer_1 — FlexRay_0	I/O O I/O — I	Slow	Medium	97	140	168
D[3]	PCR[51]	ALT0 ALT1 ALT2 ALT3	GPIO[51] CB_TX ETC[4] —	SIUL FlexRay_0 eTimer_1 —	I/O O I/O —	Slow	Symmetric	89	128	152
D[4]	PCR[52]	ALT0 ALT1 ALT2 ALT3	GPIO[52] CB_TR_EN ETC[5] —	SIUL FlexRay_0 eTimer_1 —	I/O O I/O —	Slow	Symmetric	90	129	153
D[5]	PCR[53]	ALT0 ALT1 ALT2 ALT3	GPIO[53] CS3_0 — SOUT_3	SIUL DSPI_0 — DSPI_3	I/O O — O	Slow	Medium	22	33	41
D[6]	PCR[54]	ALT0 ALT1 ALT2 ALT3	GPIO[54] CS2_0 SCK_3 SOUT_4	SIUL DSPI_0 DSPI_3 DSPI_4	I/O O I/O O	Slow	Medium	23	34	42

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
D[7]	PCR[55]	ALT0 ALT1 ALT2 ALT3 —	GPIO[55] CS3_1 — CS4_0 SIN_3	SIUL DSPI_1 — DSPI_0 DSPI_3	I/O O — O I	Slow	Medium	26	37	45
D[8]	PCR[56]	ALT0 ALT1 ALT2 ALT3	GPIO[56] CS2_1 RDY CS5_0	SIUL DSPI_1 nexus_0 DSPI_0	I/O O O O	Slow	Medium	21	32	40
D[9]	PCR[57]	ALT0 ALT1 ALT2 ALT3	GPIO[57] — TXD CS6_1	SIUL — LINFlex_1 DSPI_1	I/O — O O	Slow	Medium	15	26	34
D[10]	PCR[58]	ALT0 ALT1 ALT2 ALT3	GPIO[58] — CS0_3 —	SIUL — DSPI_3 —	I/O — I/O —	Slow	Medium	53	76	92
D[11]	PCR[59]	ALT0 ALT1 ALT2 ALT3	GPIO[59] — CS1_3 SCK_3	SIUL — DSPI_3 DSPI_3	I/O — O I/O	Slow	Medium	54	78	94
D[12]	PCR[60]	ALT0 ALT1 ALT2 ALT3 —	GPIO[60] — — DS7_1 RXD	SIUL — — DSPI_1 LINFlex_1	I/O — — O I	Slow	Medium	70	99	123
D[13]	PCR[61]	ALT0 ALT1 ALT2 ALT3	GPIO[61] — CS2_3 SOUT_3	SIUL — DSPI_3 DSPI_3	I/O — O O	Slow	Medium	67	95	119
D[14]	PCR[62]	ALT0 ALT1 ALT2 ALT3 —	GPIO[62] — CS3_3 — SIN_3	SIUL — DSPI_3 — DSPI_3	I/O — O — I	Slow	Medium	73	105	129
D[15]	PCR[63]	ALT0 ALT1 ALT2 ALT3 —	GPIO[63] — — — AN[20]	SIUL — — — ADC_0	Input Only	—	—	41	58	66

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
Port E										
E[0]	PCR[64]	ALT0 ALT1 ALT2 ALT3 —	GPIO[64] — — — AN[21]	SIUL — — — ADC_0	Input Only	—	—	46	68	80
E[1]	PCR[65]	ALT0 ALT1 ALT2 ALT3 —	GPIO[65] — — — AN[4]	SIUL — — — ADC_0	Input Only	—	—	27	39	47
E[2]	PCR[66]	ALT0 ALT1 ALT2 ALT3 —	GPIO[66] — — — AN[5]	SIUL — — — ADC_0	Input Only	—	—	32	49	57
E[3]	PCR[67]	ALT0 ALT1 ALT2 ALT3 —	GPIO[67] — — — AN[6]	SIUL — — — ADC_0	Input Only	—	—	—	40	48
E[4]	PCR[68]	ALT0 ALT1 ALT2 ALT3 —	GPIO[68] — — — AN[7]	SIUL — — — ADC_0	Input Only	—	—	—	42	50
E[5]	PCR[69]	ALT0 ALT1 ALT2 ALT3 —	GPIO[69] — — — AN[8]	SIUL — — — ADC_0	Input Only	—	—	—	44	52
E[6]	PCR[70]	ALT0 ALT1 ALT2 ALT3 —	GPIO[70] — — — AN[9]	SIUL — — — ADC_0	Input Only	—	—	—	46	54

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
E[7]	PCR[71]	ALT0 ALT1 ALT2 ALT3 —	GPIO[71] — — — AN[10]	SIUL — — — ADC_0	Input Only	—	—	—	48	56
E[8]	PCR[72]	ALT0 ALT1 ALT2 ALT3 —	GPIO[72] — — — AN[22]	SIUL — — — ADC_0	Input Only	—	—	—	59	67
E[9]	PCR[73]	ALT0 ALT1 ALT2 ALT3 —	GPIO[73] — — — AN[23]	SIUL — — — ADC_0	Input Only	—	—	—	61	69
E[10]	PCR[74]	ALT0 ALT1 ALT2 ALT3 —	GPIO[74] — — — AN[24]	SIUL — — — ADC_0	Input Only	—	—	—	63	75
E[11]	PCR[75]	ALT0 ALT1 ALT2 ALT3 —	GPIO[75] — — — AN[25]	SIUL — — — ADC_0	Input Only	—	—	—	65	77
E[12]	PCR[76]	ALT0 ALT1 ALT2 ALT3 —	GPIO[76] — — — AN[26]	SIUL — — — ADC_0	Input Only	—	—	—	67	79
E[13]	PCR[77]	ALT0 ALT1 ALT2 ALT3 —	GPIO[77] SCK_3 — — EIRQ[25]	SIUL DSPI_3 — — SIUL	I/O I/O — — I	Slow	Medium	—	117	141
E[14]	PCR[78]	ALT0 ALT1 ALT2 ALT3 —	GPIO[78] SOUT_3 — — EIRQ[26]	SIUL DSPI_3 — — SIUL	I/O O — — I	Slow	Medium	—	119	143

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
E[15]	PCR[79]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[79] — — — SIN_3 EIRQ[27]	SIUL — — — DSPI_3 SIUL	I/O — — — I I	Slow	Medium	—	121	145
Port F										
F[0]	PCR[80]	ALT0 ALT1 ALT2 ALT3 —	GPIO[80] DBG_0 CS3_3 — EIRQ[28]	SIUL FlexRay_0 DSPI_3 — SIUL	I/O O O — I	Slow	Medium	—	133	157
F[1]	PCR[81]	ALT0 ALT1 ALT2 ALT3 —	GPIO[81] DBG_1 CS2_3 — EIRQ[29]	SIUL FlexRay_0 DSPI_3 — SIUL	I/O O O — I	Slow	Medium	—	135	159
F[2]	PCR[82]	ALT0 ALT1 ALT2 ALT3	GPIO[82] DBG_2 CS1_3 —	SIUL FlexRay_0 DSPI_3 —	I/O O O —	Slow	Medium	—	137	161
F[3]	PCR[83]	ALT0 ALT1 ALT2 ALT3	GPIO[83] DBG_3 CS0_3 —	SIUL FlexRay_0 DSPI_3 —	I/O O I/O —	Slow	Medium	—	139	167
F[4]	PCR[84]	ALT0 ALT1 ALT2 ALT3	— — MDO[3] —	— — nexus_0 —	— — O —	Slow	Fast	—	4	4
F[5]	PCR[85]	ALT0 ALT1 ALT2 ALT3	— — MDO[2] —	— — nexus_0 —	— — O —	Slow	Fast	—	5	13
F[6]	PCR[86]	ALT0 ALT1 ALT2 ALT3	GPIO[86] — MDO[1] —	SIUL — nexus_0 —	I/O — O —	Slow	Fast	—	8	16

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
F[7]	PCR[87]	ALT0 ALT1 ALT2 ALT3	GPIO[87] — MCKO —	SIUL — nexus_0 —	I/O — O —	Slow	Fast	—	19	27
F[8]	PCR[88]	ALT0 ALT1 ALT2 ALT3	GPIO[88] — MSEO1 —	SIUL — nexus_0 —	I/O — O —	Slow	Fast	—	20	28
F[9]	PCR[89]	ALT0 ALT1 ALT2 ALT3	GPIO[89] — MSEO0 —	SIUL — nexus_0 —	I/O — O —	Slow	Fast	—	23	31
F[10]	PCR[90]	ALT0 ALT1 ALT2 ALT3	GPIO[90] — EVTO —	SIUL — nexus_0 —	I/O — O —	Slow	Fast	—	24	32
F[11]	PCR[91]	ALT0 ALT1 ALT2 ALT3	GPIO[91] EVTI — —	SIUL nexus_0 — —	I/O I — —	Slow	Medium	—	25	33
F[12]	PCR[92]	ALT0 ALT1 ALT2 ALT3	GPIO[92] ETC[3] — —	SIUL eTimer_1 — —	I/O I/O — —	Slow	Medium	—	106	130
F[13]	PCR[93]	ALT0 ALT1 ALT2 ALT3	GPIO[93] ETC[4] — —	SIUL eTimer_1 — —	I/O I/O — —	Slow	Medium	—	112	136
F[14]	PCR[94]	ALT0 ALT1 ALT2 ALT3	GPIO[94] TXD — —	SIUL LINFlex_1 — —	I/O O — —	Slow	Medium	—	115	139
F[15]	PCR[95]	ALT0 ALT1 ALT2 ALT3 —	GPIO[95] — — — RXD	SIUL — — — LINFlex_1	I/O — — — I	Slow	Medium	—	113	137

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2],[3]	Functions	Peripheral[4]	I/O direction[5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
Port G										
G[0]	PCR[96]	ALT0 ALT1 ALT2 ALT3 —	GPIO[96] F[0] — — EIRQ[30]	SIUL FCCU — — SIUL	I/O O — — I	Slow	Medium	—	38	46
G[1]	PCR[97]	ALT0 ALT1 ALT2 ALT3 —	GPIO[97] F[1] — — EIRQ[31]	SIUL FCCU — — SIUL	I/O O — — I	Slow	Medium	—	141	173
G[2]	PCR[98]	ALT0 ALT1 ALT2 ALT3 —	GPIO[98] — — — SIN_4	SIUL — — — DSPI_4	I/O — — — I	Slow	Medium	—	102	126
G[3]	PCR[99]	ALT0 ALT1 ALT2 ALT3	GPIO[99] — SOUT_4 —	SIUL — DSPI_4 —	I/O — O —	Slow	Medium	—	104	128
G[4]	PCR[100]	ALT0 ALT1 ALT2 ALT3	GPIO[100] — SCK_4 —	SIUL — DSPI_4 —	I/O — I/O —	Slow	Medium	—	100	124
G[5]	PCR[101]	ALT0 ALT1 ALT2 ALT3	GPIO[101] — CS0_4 —	SIUL — DSPI_4 —	I/O — I/O —	Slow	Medium	—	85	103
G[6]	PCR[102]	ALT0 ALT1 ALT2 ALT3	GPIO[102] — CS1_4 —	SIUL — DSPI_4 —	I/O — O —	Slow	Medium	—	98	122
G[7]	PCR[103]	ALT0 ALT1 ALT2 ALT3	GPIO[103] — CS2_4 —	SIUL — DSPI_4 —	I/O — O —	Slow	Medium	—	83	101

Table 7. Pin muxing[1] (continued)

Port pin	PCR register	Alternate function[2], [3]	Functions	Peripheral [4]	I/O direction [5]	Pad speed[6]		Pin		
						SRC = 0	SRC = 1	LQFP 100	LQFP 144	LQFP 176[7]
G[8]	PCR[104]	ALT0 ALT1 ALT2 ALT3 —	GPIO[104] — CS3_4 —	SIUL — DSPI_4 —	I/O — O —	Slow	Medium	—	81	97
G[9]	PCR[105]	ALT0 ALT1 ALT2 ALT3 —	GPIO[105] — — — RXD	SIUL — — — FlexCAN_1	I/O — — — I	Slow	Medium	—	79	95
G[10]	PCR[106]	ALT0 ALT1 ALT2 ALT3	GPIO[106] — TXD —	SIUL — FlexCAN_1 —	I/O — O —	Slow	Medium	—	77	93
G[11]	PCR[107]	ALT0 ALT1 ALT2 ALT3	GPIO[107] — — —	SIUL — — —	I/O — — —	Slow	Medium	—	75	91

1. This table concerns Enhanced Full-featured version. Please refer to “SPC56xP60/54 device configuration difference” table for difference between Enhanced Full-featured, Full-featured, and Airbag configuration.
2. ALT0 is the primary (default) function for each port after reset.
3. Alternate functions are chosen by setting the values of the PCR[PA] bitfields inside the SIU module. PCR[PA] = 00 → ALT0; PCR[PA] = 01 → ALT1; PCR[PA] = 10 → ALT2; PCR[PA] = 11 → ALT3. This is intended to select the output functions; to use one of the input-only functions, the PCR[IBE] bit must be written to ‘1’, regardless of the values selected in the PCR[PA] bitfields. For this reason, the value corresponding to an input only function is reported as “—”.
4. Module included on the MCU.
5. Multiple inputs are routed to all respective modules internally. The input of some modules must be configured by setting the values of the PSMI[PADSEL*x*] bitfields inside the SIUL module.
6. Programmable via the SRC (Slew Rate Control) bits in the respective Pad Configuration Register.
7. LQFP176 available only as development package.
8. Weak pull down during reset.

3.5 CTU / ADCs / eTimers / DSPI / FlexRay connections

Figure 6 shows the interconnections between the CTU, ADCs, eTimers, DSPI and FlexRay.

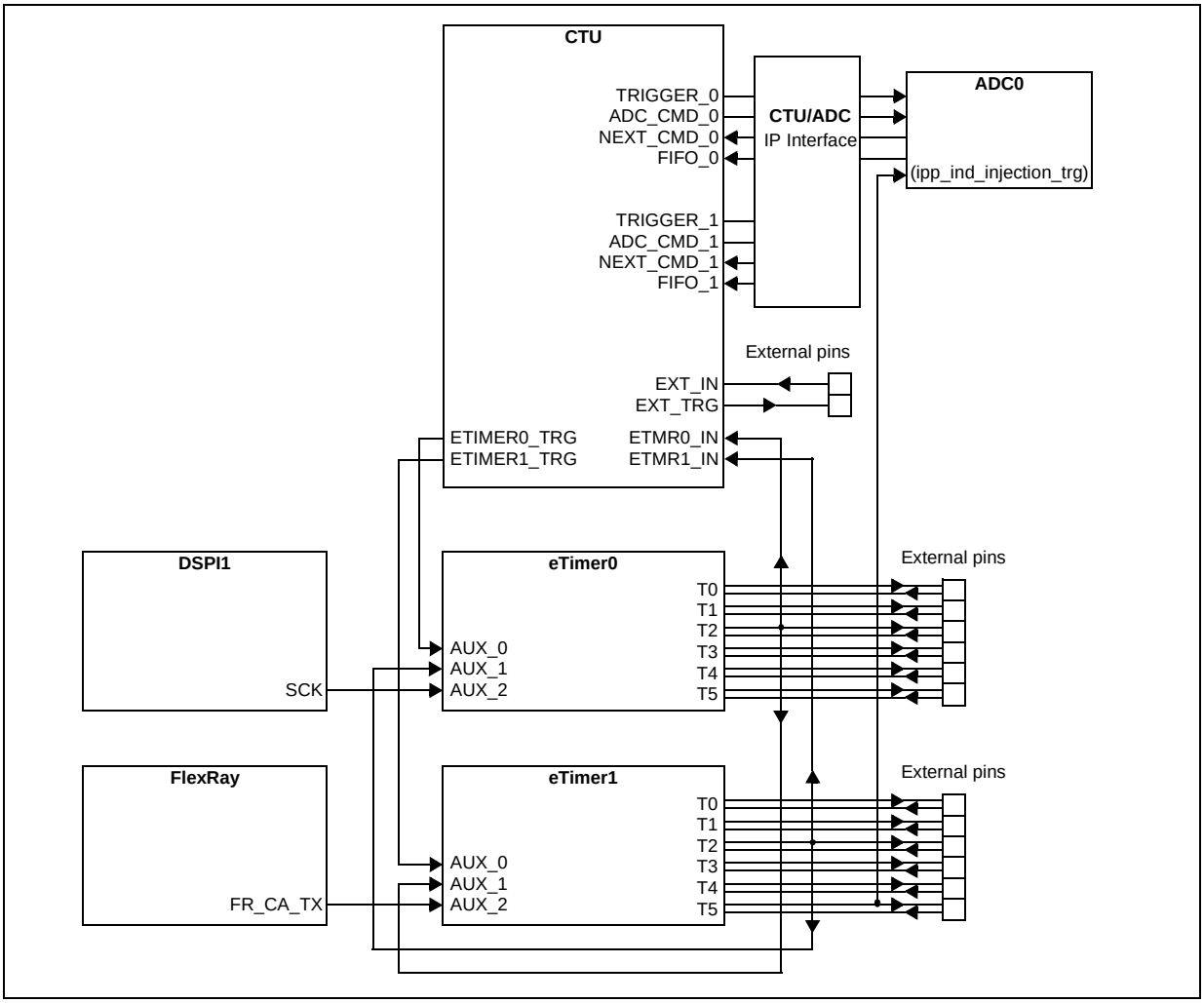


Figure 6. CTU / ADCs / eTimers / DSPI / FlexRay connections

Table 8. CTU / ADCs / eTimers / DSPI / FlexRay connections

Source module (Signal)	Target module (Signal)	Comment
eTimer_0(T2)	CTU(ETMR0_IN)	—
eTimer_0(T2)	eTimer_1(AUX_1)	—
eTimer_1(T2)	CTU(ETMR1_IN)	—
eTimer_1(T2)	eTimer_0(AUX_1)	—
eTimer_1(T5)	ADC_0	A/D conversion injection request signal (for non CTU mode of operation).
CTU(ETIMER0_TRG)	eTimer_0(AUX_0)	—

Table 8. CTU / ADCs / eTimers / DSPI / FlexRay connections (continued)

Source module (Signal)	Target module (Signal)	Comment
CTU(ETIMER1_TRG)	eTimer_1(AUX_0)	—
CTU(TRIGGER_0)	ADC_0 (Through CTU/ADC IP Interface)	—
CTU(ADC_CMD_0)	ADC_0 (Through CTU/ADC IP Interface)	16-bits signal
CTU(TRIGGER_1)	Virtual ADC_1[1] (Through CTU/ADC IP Interface)	—
CTU(ADC_CMD_1)	Virtual ADC_1[1] (Through CTU/ADC IP Interface)	16-bits signal
CTU(EXT_TGR)	SIU lite	—
ADC_0(EOC)	CTU(NEXT_CMD_0) (Through CTU/ADC IP Interface)	End Of Conversion should be used as next command request signal
ADC_0	CTU(FIFO_0) (Through CTU/ADC IP Interface)	18-bits signal
Virtual ADC_1[1](EOC)	CTU(NEXT_CMD_1) (Through CTU/ADC IP Interface)	End Of Conversion should be used as next command request signal
Virtual ADC_1[1]	CTU(FIFO_1) (Through CTU/ADC IP Interface)	18-bits signal
SIU lite	CTU(EXT_IN)	The same GPIO pin as used for CTU (EXT_IN)
DSPI_1(SCK)	eTimer_0(AUX_2)	—
FlexRAY(FR_CA_TX)	eTimer_1(AUX_2)	—

1. ADC_1 commands are translated into ADC_0 commands.

4 Clock Description

This chapter describes the clock architectural implementation for SPC56xP60/54.

The following clock related modules are implemented on the SPC56xP60/54:

- Clock, Reset, and Mode Handling
  - Clock Generation Module (CGM) (see *Chapter 5: Clock Generation Module (MC_CGM)*)
  - Reset Generation Module (RGM) (see *Chapter 8: Reset Generation Module (MC_RGM)*)
  - Mode Entry Module (ME) (see *Chapter 6: Mode Entry Module (MC_ME)*)
- High Frequency Oscillator (XOSC) (see *Section 4.8: XOSC external crystal oscillator*)
- High Frequency RC Oscillator (IRC) (see *Section 4.7: IRC 16 MHz internal RC oscillator (RC_CTL)*)
- FMPLL (FMPLL_0) (see *Section 4.9: Frequency Modulated Phase Locked Loop (FMPLL)*)
- CMU (CMU_0) (see *Section 4.10: Clock Monitor Unit (CMU)*)
- CMU (CMU_1) (see *Section 4.10: Clock Monitor Unit (CMU)*)
- Periodic Interrupt Timer (PIT) (see *Chapter 32: Periodic Interrupt Timer (PIT)*)
- System Timer Module (STM_0) (see *Chapter 33: System Timer Module (STM)*)
- System Timer Module (STM_1) (see *Chapter 33: System Timer Module (STM)*)
- Software Watchdog Timer (SWT_0) (see *Section 29.3: Software Watchdog Timer (SWT)*)
- Software Watchdog Timer (SWT_1) (see *Section 29.3: Software Watchdog Timer (SWT)*)

4.1 Clock architecture

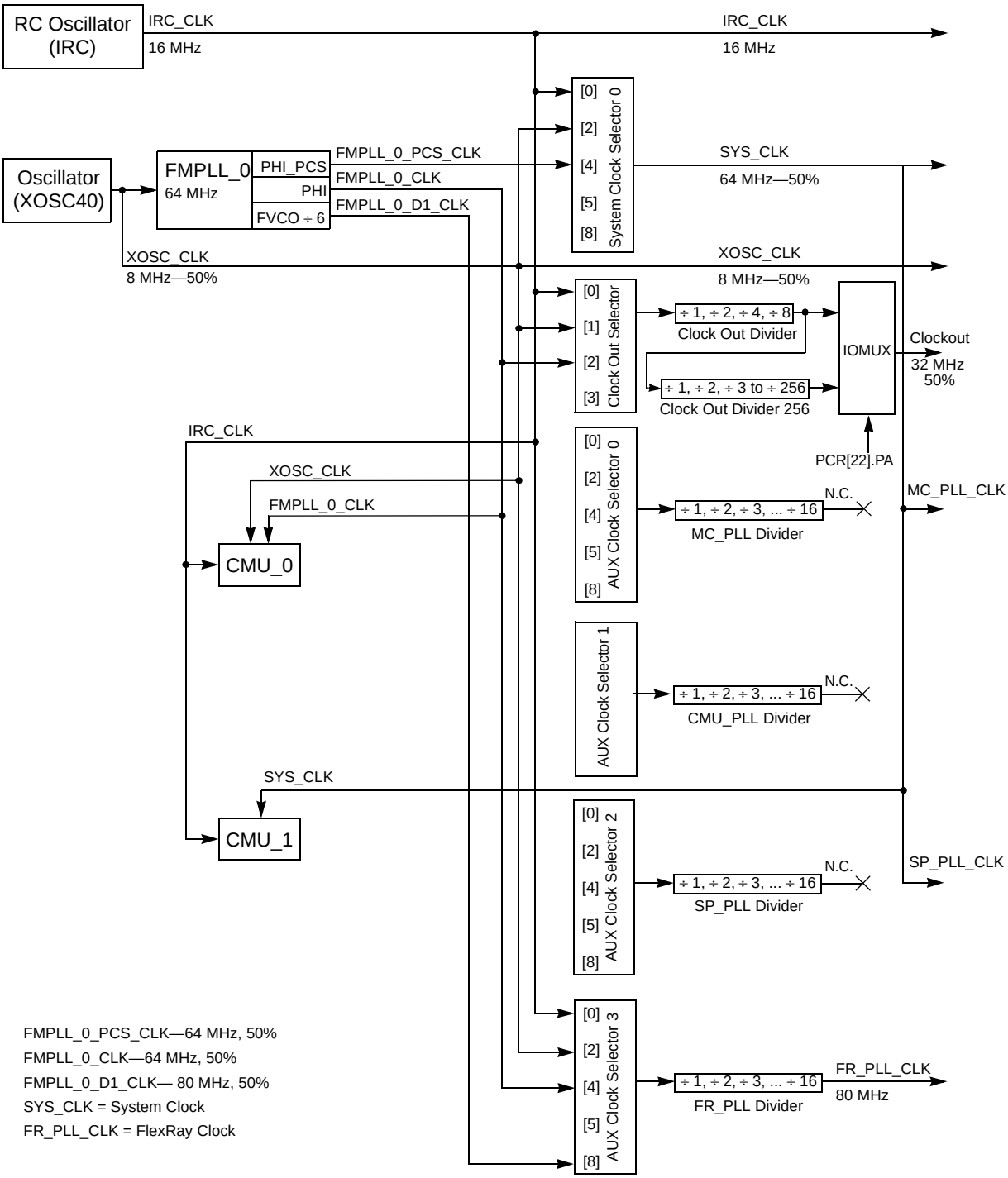
The system and peripheral clocks are generated from three sources:

- IRC—internal RC oscillator clock
- XOSC—oscillator clock
- FMPLL_0 clock output

The clock architecture is shown in *Figure 7*, *Figure 8*, and *Figure 9*.

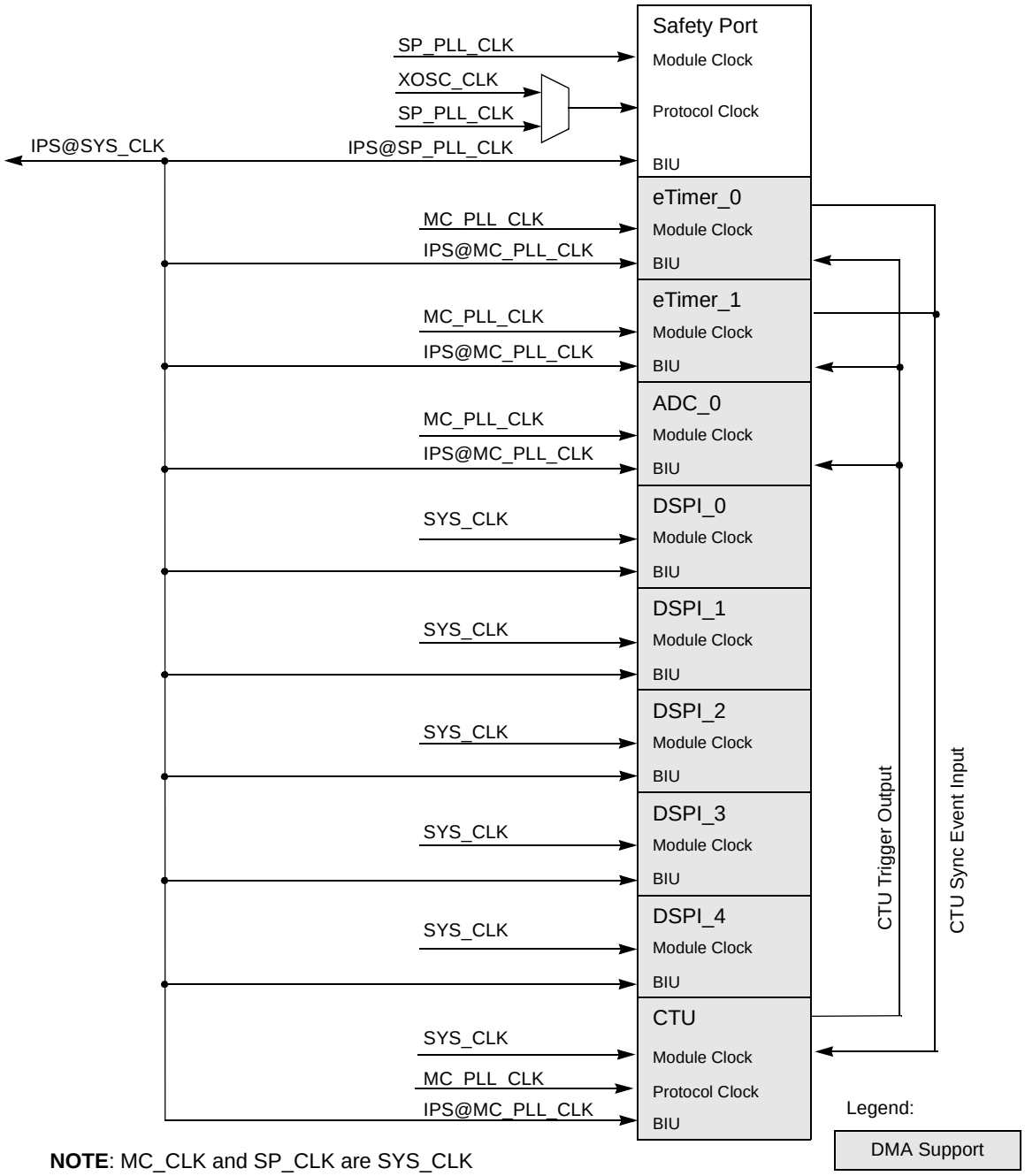
The frequencies shown in *Figure 7* represent only one possible setting.

Note: MC_PLL_CLK and SP_PLL_CLK are SYS_CLK.



NOTE: FlexRay protocol clock does not support IRC as a clock source.
MC_CLK and SP_CLK are SYS_CLK

Figure 7. SPC56xP60/54 system clock generation



NOTE: MC_CLK and SP_CLK are SYS_CLK

Figure 8. SPC56xP60/54 system clock distribution Part A

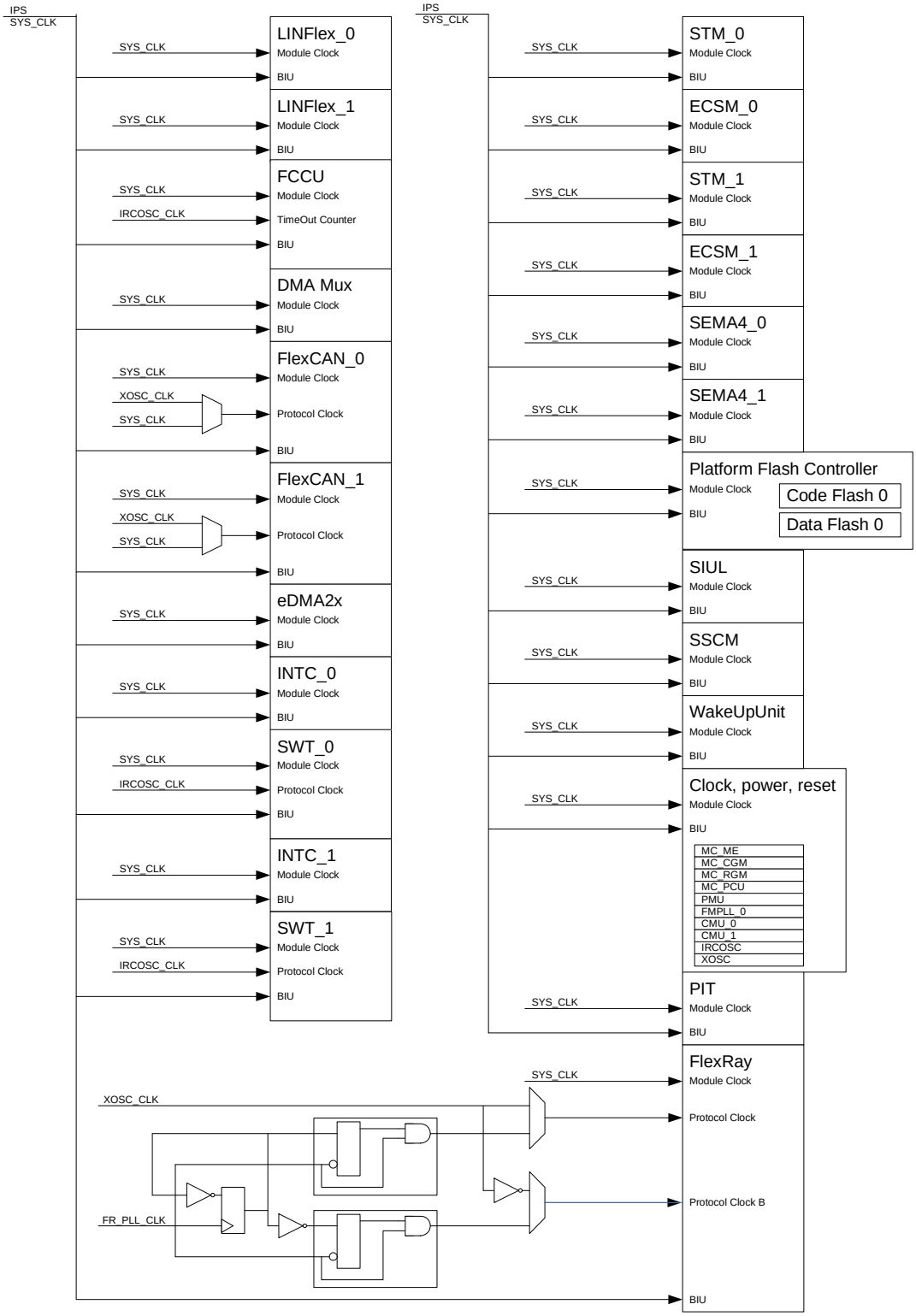


Figure 9. SPC56xP60/54 system clock distribution Part B

4.2 Available clock domains

This section describes the various clock domains available on SPC56xP60/54.

4.2.1 FMPLL input reference clock

The input reference clock for FMPLL_0 is always the external crystal oscillator clock (XOSC).

4.2.2 Clock selectors

System clock selector 0 for SYS_CLK

The system clock selector 0 selects the clock source for the system clock (SYS_CLK) from clock signals:

- Internal RC oscillator clock (IRC)
- Progressive output clock of FMPLL_0
- Directly from the oscillator clock (XOSC)

Its behavior is configured via software through ME_*x*_MC register of the ME module.

When the standard boot from internal flash is selected via the boot configuration pins, the clock source for the system clock (SYS_CLK) after reset (DRUN mode) is the internal RC oscillator (IRC).

4.2.3 Auxiliary Clock Selector 0

There is no Auxiliary Clock selector 0 present on SPC56xP60/54 device, but to maintain the software compatibility, corresponding register in MC_CGM (CGM_AC0_SC) has been implemented through which user can select any clock source from the given auxiliary clock sources. As there is no auxiliary clock, all the auxiliary clock sources have been tied to ‘0’.

4.2.4 Auxiliary Clock Selector 1

There is no Auxiliary Clock selector 1 present on SPC56xP60/54 device, but to maintain the software compatibility, corresponding register in MC_CGM (CGM_AC1_SC) has been implemented through which user can select any clock source from the given auxiliary clock sources. As there is no auxiliary clock, all the auxiliary clock sources have been tied to ‘0’.

4.2.5 Auxiliary Clock Selector 2

There is no Auxiliary Clock selector 2 present on SPC56xP60/54 device, but to maintain the software compatibility, corresponding register in MC_CGM (CGM_AC2_SC) has been implemented through which user can select any clock source from the given auxiliary clock sources. As there is no auxiliary clock, all the auxiliary clock sources have been tied to ‘0’.

Auxiliary Clock Selector 3 for FR_PLL divider (FlexRay clock)

Software uses the clock output selector to select one of the following clock sources:

- Internal RC oscillator clock (IRC)
- Progressive output clock of FMPLL_0
- Directly from the oscillator clock (XOSC)

The default clock source after reset is the IRC, but the default status is disabled.

The clock output can be enabled or disabled by software through the CGM module (CGM_OC_EN and CGM_OCDS_SC registers).

4.2.6 Auxiliary clock dividers

The auxiliary clock divider is implemented to generate the FR_PLL_CLK and supports these divide options: ÷ 1, ÷ 2, ÷ 3, ÷ 4, ÷5 , ÷ 6, ÷ 7, ÷ 8, ÷ 9, ÷ 10, ÷ 11, ÷ 12, ÷ 13, ÷ 14, ÷ 15.

The clock divider is configured by CGM_AC3_DC0 register of the CGM module.

To maintain the software compatibility, one divider corresponding to every missing auxiliary clock has been implemented. Corresponding registers have been implemented in MC_CGM which can be accessed by user but have no impact in device. These registers are CGM_AC0_DC0, CGM_AC1_DC0, and CGM_AC2_DC0

4.2.7 External clock divider

The output clock divider provides a nominal 50% duty cycle clock and allows the selected output clock source to be divided with these divide options:

- ÷ 1, ÷ 2, ÷ 4, ÷ 8
- ÷ 1, ÷ 2, ÷ 3 to ÷ 256

Output Clock Division 256 Select Register (CLK_DIV_256)

Figure 10. Output Clock Division 256 Select Register (CLK_DIV_256)

Address 0xC3FE_0160 Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DIV								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to select by which factor the selected output source is divided before being delivered at the output clock.

Table 9. Output Clock Division 256 Select Register (CLK_DIV_256) Field Descriptions

Field	Description
DIV	**Output Clock Division 256 Select** 0x00 output selected Output Clock without division 0x01 output selected Output Clock divided by 2 0x10 output selected Output Clock divided by 3 ... 0xFF output selected Output Clock divided by 256

4.3 Alternate module clock domains

This section lists the different clock domains for each module. If not otherwise noted, all modules on the SPC56xP60/54 device are clocked on the SYS_CLK.

4.3.1 FlexCAN clock domains

The FlexCAN modules have two distinct software controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be either the system clock (SYS_CLK) or a direct feed from the oscillator pin XOSC_CLK. The logic in the second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures very low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR. *Figure 449* shows the two clock domains in the FlexCAN modules.

Refer to *Chapter 24: FlexCAN* for more information on the FlexCAN modules.

4.3.2 FlexRay clock domains

The FlexRay block has two distinct clock domains. The first clock domain (Module Clock or CHI clock) is always derived from the SYS_CLK clock.

The source for the second clock domain can either be the FR_PLL_CLK clock or the XOSC_CLK clock. The logic in the second clock domain controls the FlexRay interface pins (Protocol Clock or PE clock).

4.3.3 SWT clock domains

The SWT module has two distinct clock domains. The first clock domain (Module Clock) is always supplied from the SYS_CLK. This clock domain includes the register interface.

The source for the second clock domain (Protocol Clock) is always the IRC generated by the internal RC oscillator.

4.3.4 Nexus Message Clock (MCKO)

The Nexus message clock (MCKO) divider can be programmed to divide the system clock by one, two, four or eight based on the MCKO_DIV bit field in the port configuration register (PCR) in the Nexus port controller (NPC). The reset value of the MCKO_DIV selects an MCKO clock frequency one half of the system clock frequency. The MCKO divider is configured by writing to the NPC through the JTAG port. Refer to *Chapter 38: Nexus Port Controller (NPC)* for more information.

4.3.5 Cross Triggering Unit (CTU) clock domains

The CTU module has two distinct clock domains. The first clock domain (Module Clock) is supplied from the SYS_CLK. This clock domain includes the Command Buffer logic.

The source for the second clock domain (Protocol Clock) is the MC_PLL_CLK. The logic in the Protocol Clock domain controls the CTU interface pins to the eTimer module and the ADC module.

4.3.6 Peripherals

eTimer_0 clock domain

The eTimer_0 module has only one clock domain. The eTimer_0 module is clocked from the MC_PLL_CLK.

eTimer_1 clock domain

The eTimer_1 module has only one clock domain. The eTimer_1 module is clocked from the MC_PLL_CLK.

ADC_0 clock domain

The ADC_0 module has only one clock domain. The ADC_0 module is clocked from the MC_PLL_CLK.

Safety Port clock domains

The Safety Port module has two distinct software-controlled clock domains. The first clock domain (Module Clock) is always supplied from the SP_PLL_CLK. The source for the second clock domain (Protocol Clock) can either be the SP_PLL_CLK or the XOSC_CLK.

The user must ensure that the frequency of the first clock domain (Module Clock) clocked from the MC_PLL_CLK is always the same or greater than the clock selected for the second clock domain (Protocol Clock).

Core_1 switching off

The Core_1 can be switched off (CORE_1 HALT ENTRY) by the Core_0 writing into the ME_PCTLx register, MC_ME asserts the signal zcor_halt and this signal from platform is used to gate the clock to Core_1. The Core_1 can be switched on (CORE_1 HALT EXIT) by the Core_0 writing into the ME_PCTLx register, Core_1 clock is enabled by MC_ME. Together with Core_1 clock, is switched off or switched on also the clock of all ON-platform_1 peripherals (INTC, SWT, STM, ECSM, SEMA4).

ME_PCTL number for Core_1 is 127.

4.4 Clock behavior in STOP and HALT mode

In this section the term “resume” is used to describe the transition from STOP and HALT mode back to a RUN mode.

The SPC56xP60/54 supports the STOP and the HALT modes. These two modes allow to put the device into a power saving mode with the configuration options defined in the ME module.

The following constraints are applied on SPC56xP60/54 to guarantee that in all modes of operation a resume from STOP or HALT mode is always possible without the need to reset:

- STOP and HALT mode:
  - SIUL clock is not gateable
  - FCCU clock must not be gateable
  - SIUL filter for external interrupt capable pins is always clocked with IRC
  - Resume via interrupt that can be generated by any peripheral that clock is not gated
  - Resume via $\overline{\text{NMI}}$ pin is always possible if once enabled after reset (no software configuration that could block resume afterwards)
- STOP mode:
  - IRC can NOT be switched off
  - The System Clock Selector 0 is switched to the IRC and therefore the SYS_CLK is feed by the IRC signal
  - Resume via external interrupt pin is always possible (if not masked)
- HALT mode:
  - The output of the System Clock Selector 0 can only be switched to a running clock input
  - Resume via external interrupt pin is always possible (if not masked) and IRC is not switched off

4.5 Software controlled power management/clock gating

Some of the IP modules on this device support software controlled power management/clock gating whereby the application software can disable the non-memory-mapped portions of the modules by writing to module disable (MDIS) bits in registers within the modules. The memory-mapped portions of the modules are clocked by the system clock when they are being accessed. The NPC can be configured to disable the MCKO signal when there are no Nexus messages pending.

The modules that implemented software controlled power management/clock gating are listed in *Table 10* along with the registers and bits that disable each module. The software controlled clocks are enabled when the MCU comes out of reset.

Table 10. Software controlled power management/clock gating support

Module name	Register name	Bit names
NPC	NPC_PCR	MCKO_EN, MCKO_GT(1)

1. Refer to *Chapter 38: Nexus Port Controller (NPC).*

4.6 System clock functional safety

This section shows the SPC56xP60/54 modules used to detect clock failures:

- The Clock Monitoring Unit (CMU_0) monitors the clock frequency of the FMPLL_0 and the XOSC signal against the IRC and provides clock out of range information about the monitored clock signals.
- FMPLL_0 provides a signal that indicates a loss of lock. Each loss of lock signal is sent to the CGM module.

Upon the detection of one of the above mentioned failures, the SPC56xP60/54 device either asserts a reset, generates an interrupt, or sends the device into the SAFE state.

The reaction to each of the clock failures and system parameters (like active clocks and SYS_CLK clock source) that become active in SAFE state are under software control and can be configured in the ME module.

4.7 IRC 16 MHz internal RC oscillator (RC_CTL)

The IRC output frequency can be trimmed using RCTRIM bits. After a power-on reset, the IRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at RC_CTL[RCTRIM], and this field shows a value of zero. Therefore, be aware that the RC_CTL[RCTRIM] field does not reflect the current trim value until you have written to it. Pay particular attention to this feature when you initiate a read-modify-write operation on RC_CTL, because a RCTRIM value of zero may be unintentionally written back and this may alter the IRC frequency. In this case, you should calibrate the IRC using the CMU.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from -32 to 31. As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to device datasheet for average frequency variation of the trimming step.

Figure 11. RC Control register (RC_CTL)

Address: 0xC3FE_0060 (Base + 0x0000)

Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	RCTRIM[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11. RC_CTL field descriptions

Field	Description
RCTRIM[5:0]	Main RC trimming bits

4.8 XOSC external crystal oscillator

The external crystal oscillator (XOSC) operates in the range of 4 MHz to 40 MHz. The XOSC digital interface contains the control and status registers accessible for the external crystal oscillator.

Main features are:

- Oscillator clock available interrupt
- Oscillator bypass mode

4.8.1 Functional description

The crystal oscillator circuit includes an internal oscillator driver and an external crystal circuitry. The XOSC provides an output clock to the PLL or it is used as a reference clock to specific modules depending on system needs.

The crystal oscillator can be controlled by the ME:

- Control by ME module. The OSCON bit of the ME_xxx_MCRs controls the powerdown of oscillator based on the current device mode while S_OSC of ME_GS register provides the oscillator clock available status.

After system reset, the oscillator is put to power down state and software has to switch on when required. Whenever the crystal oscillator is switched on from off state, OSCCNT counter starts and when it reaches the value EOCV[7:0] × 512, oscillator clock is made available to the system. Also an interrupt pending bit I_OSC of OSC_CTL register is set. An interrupt will be generated if the interrupt mask bit M_OSC is set.

The oscillator circuit can be bypassed by setting OSC_CTL[OSCBYP]. This bit can only be set by the software. System reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as external clock applied on EXTAL pin and the oscillator status is forced to ‘1’. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 12 shows the truth table of different configurations of oscillator.

Table 12. Crystal oscillator truth table

ENABLE	BYP	XTALIN	EXTAL	CK_OSCM	XOSC Mode
0	0	No crystal, High Z	No crystal, High Z	0	Power down, IDDQ
x	1	x	Ext clock	EXTAL	Bypass, XOSC disabled
1	0	Crystal	Crystal	EXTAL	Normal, XOSC enabled

4.8.2 Register description

Table 13. OSC_CTL memory map

Offset from OSC_CTL_BASE (0xC3FE_0000)	Register	Location
0x0000	OSC_CTL—Oscillator control register	*on page 112*
0x0004–0x000F	Reserved	

Figure 12. Crystal Oscillator Control register (OSC_CTL)

Address: 0xC3FE_0000 (Base + 0x0000) Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OSC BYP	0	0	0	0	0	0	0	EOCV[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M_ OSC	0	0	0	0	0	0	0	I_ OSC	0	0	0	0	0	0	0
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14. OSC_CTL field descriptions

Field	Description
OSCBYP	Crystal Oscillator bypass This bit specifies whether the oscillator should be bypassed or not. Software can only set this bit. System reset is needed to reset this bit. 0: Oscillator output is used as root clock. 1: EXTAL is used as root clock.
EOCV[7:0]	End of Count Value These bits specify the end of count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state. This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV[7:0]*512, oscillator available interrupt request is generated. The reset value of this field depends on the device specification. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
M_OSC	Crystal oscillator clock interrupt mask 0: Crystal oscillator clock interrupt masked 1: Crystal oscillator clock interrupt enabled
I_OSC	Crystal oscillator clock interrupt This bit is set by hardware when OSCCNT counter reaches the count value EOCV[7:0]*512. It is cleared by software by writing 1. 0: No oscillator clock interrupt occurred 1: Oscillator clock interrupt pending

4.9 Frequency Modulated Phase Locked Loop (FMPLL)

4.9.1 Introduction

This section describes the features and functions of the FMPLL module implemented in SPC56xP60/54.

4.9.2 Overview

The FMPLL enables the generation of high speed system clocks from a common 4–40 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor and output clock divider ratio are all software configurable.

The FMPLL block diagram is shown in *Figure 13*.

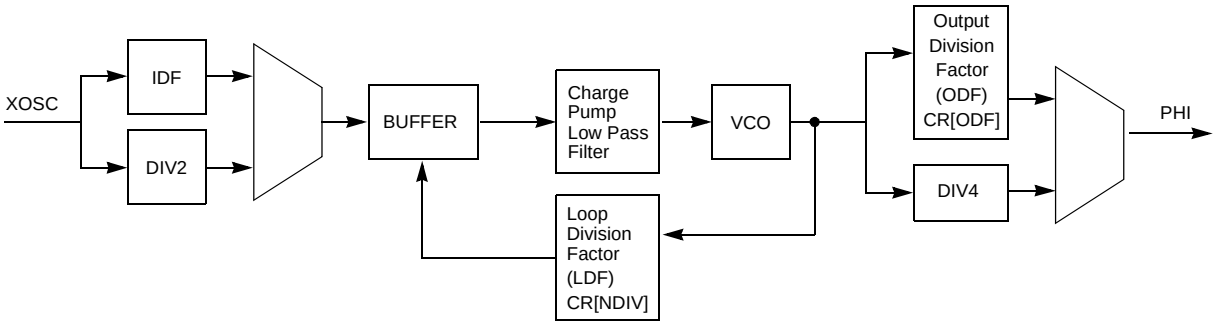


Figure 13. FMPLL block diagram

4.9.3 Features

The FMPLL has the following major features:

- Input clock frequency 4–40 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated PLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth
  - ±0.25% to ±4% deviation from center spread frequency
  - –0.5% to +8% deviation from down spread frequency
  - Programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation
- 4 available modes
  - Normal mode
  - Progressive clock switching
  - Normal Mode with SSCG
  - Powerdown mode

4.9.4 Memory map

Table 15 shows the memory map locations. Addresses are given as offsets of the module base address.

Table 15. FMPLL memory map

Offset from ME_CGM_BASE[1] FMPLL_0: 0xC3FE_00A0	Register	Location
0x0000	CR—Control Register	*on page 115*
0x0004	MR—Modulation register	*on page 117*
0x0004–0x000F	Reserved	

1. FMPLL_x are mapped through the ME_CGM Register Slot

4.9.5 Register description

The PLL operation is controlled by two registers. Those registers can only be written in supervisor mode.

Control Register (CR)

Figure 14. Control Register (CR)

Address: Base + 0x0000
FMPLL_0 = 0xC3FE_00A0

Access: Supervisor read/write
User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	IDF[3:0]				ODF[1:0]		0	NDIV[6:0]						
W																
Reset	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	en_pll_sw	0	unlock_once	0	i_lock	s_lock	pll_fail_mask	pll_fail_flag	1
W												w1c			w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 16. CR field descriptions

Field	Description
IDF[3:0]	Input Division Factor The value of this field sets the PLL input division factor. 0000: Divide by 1 0001: Divide by 2 0010: Divide by 3 0011: Divide by 4 0100: Divide by 5 0101: Divide by 6 0110: Divide by 7 0111: Divide by 8 1000: Divide by 9 1001: Divide by 10 1010: Divide by 11 1011: Divide by 12 1100: Divide by 13 1101: Divide by 14 1110: Divide by 15 1111: Clock Inhibit
ODF[1:0]	Output Division Factor The value of this field sets the PLL output division factor. 00: Divide by 2 01: Divide by 4 10: Divide by 8 11: Divide by 16
NDIV[6:0]	Loop Division Factor The value of this field sets the PLL loop division factor. 0000000–0011111: Reserved 0100000: Divide by 32 0100001: Divide by 33 0100010: Divide by 34 ... 1011111: Divide by 95 1100000: Divide by 96 1100001–1111111: Reserved
en_pll_sw	This bit is used to enable progressive clock switching. After the PLL locks, the PLL output initially is divided by 8, and then progressively decreases until it reaches divide-by-1. Note: The PLL output should not be used if a non-changing clock is needed, such as for serial communications, until the division has finished. 0: Progressive clock switching disabled 1: Progressive clock switching enabled
unlock_once	This bit is a sticky indication of PLL loss of lock condition. Unlock_once is set when the PLL loses lock. Whenever the PLL reacquires lock, unlock_once remains set. unlock_once is cleared after a POR event.

Table 16. CR field descriptions (continued)

Field	Description
i_lock	This bit is set by hardware whenever there is a lock/unlock event.It is cleared by software, writing 1.
s_lock	This bit indicates whether the PLL has acquired lock. 0: PLL unlocked 1: PLL locked
pll_fail_mask	This bit masks the pll_fail output. 0: pll_fail not masked 1: pll_fail masked
pll_fail_flag	This bit is asynchronously set by hardware whenever a loss of lock event occurs while PLL is switched on. It is cleared by software, writing 1.

Modulation Register (MR)

Figure 15. Modulation Register (MR)

Address: Base + 0x0004

Access: Supervisor read/write
User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STRB_BYPASS	0	SPRD_SEL	MOD_PERIOD												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FM_EN	INC_STEP														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 17. MR field descriptions

Field	Description
STRB_BYPASS	Strobe bypass The STRB_BYPASS signal bypasses the STRB signal used inside the PLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0: STRB latches the PLL modulation control bits. 1: STRB is bypassed. In this case, the control bits need to be static. The control bits must be changed only when PLL is in power down mode.
SPRD_SEL	Spread type selection The SPRD_SEL bit selects the spread type in Frequency Modulation mode. 0: Center spread 1: Down spread

Table 17. MR field descriptions (continued)

Field	Description
MOD_PERIOD	Modulation period The MOD_PERIOD field is the binary equivalent of the value modperiod derived from following formula: $modperiod = \frac{f_{ref}}{4 \times f_{mod}}$ where: *fref:* represents the frequency of the feedback divider *fmod*: represents the modulation frequency
FM_EN	Frequency modulation enable The FM_EN bit enables the frequency modulation. 0: Frequency Modulation disabled 1: Frequency Modulation enabled
INC_STEP	Increment step The INC_STEP field is the binary equivalent of the value incstep derived from following formula: $incstep = round\left(\frac{(2^{15} - 1) \times md \times MDF}{100 \times 5 \times MODPERIOD}\right)$ where: *md*: represents the peak modulation depth in percentage (Center spread — pk-pk = ±md, Downspread — pk-pk = –2 × md) *MDF*: represents the nominal value of loop divider (NDIV in PLL Control Register).

4.9.6 Functional description

Normal mode

In Normal mode, the PLL inputs are driven by the Control Register (CR). This means that when the PLL is locked, the PLL output clock (PHI) is derived from the reference clock (XOSC) through this relationship:

Equation 1

$$phi = \frac{xosc \cdot ldf}{idf \cdot odf}$$

where the value of *idf* (Input Division Factor), *ldf* (Loop Division Factor), and *odf* (Output Division Factor) are set in the CR as shown in *Table 16*. *idf* and *odf* are specified in the IDF and ODF bitfields, respectively; *ldf* is specified in the NDIV bitfield.

Progressive clock switching

Progressive clock switching allows to switch system clock to PLL output clock stepping through different division factors. This means that the current consumption gradually increases and, in turn, voltage regulator response is improved.

This feature can be enabled by programming bit en_pll_sw in the CR. Then, when the PLL is selected as the system clock, the output clock progressively increases its frequency as shown in *Table 18*.

Table 18. Progressive clock switching on pll_select rising edge

Number of PLL output clock cycles	ck_pll_div frequency (PLL output frequency)
8	(ck_pll_out frequency) ÷ 8
16	(ck_pll_out frequency) ÷ 4
32	(ck_pll_out frequency) ÷ 2
onward	(ck_pll_out frequency)

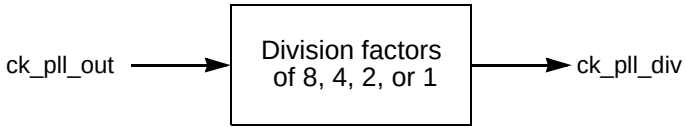


Figure 16. Progressive clock switching scheme

Normal Mode with frequency modulation

The FMPLL default mode is without frequency modulation enabled. When frequency modulation is enabled, however, two parameters must be set to generate the desired level of modulation: the PERIOD, and the STEP. The modulation waveform is always a triangle wave and its shape is not programmable.

Frequency modulation is activated as follows:

1. Configure the FM modulation characteristics: MOD_PERIOD, INC_STEP.
2. Enable the FM modulation by programming bit SSCG_EN of the MR to ‘1’. FM modulated mode can be enabled only when PLL is in lock state.

There are two ways to latch these values inside the FMPLL, depending on the value of bit STRB_BYPASS in the MR.

If STRB_BYPASS is low, the modulation parameters are latched in the PLL only when the strobe signal goes high. The strobe signal is automatically generated in the FMPLL when the modulation is enabled (SSCG_EN goes high) if the PLL is locked (s_lock = 1) or when the modulation has been enabled (SSCG_EN = 1) and PLL enters in lock state (s_lock goes high).

If STRB_BYPASS is high, the strobe signal is bypassed. In this case, control bits (MOD_PERIOD[12:0], INC_STEP[14:0], SPREAD_CONTROL) must be changed only when the PLL is in power down mode.

The modulation depth in % is

Equation 2

$$\text{ModulationDepth} = \left(\frac{100 \times 5 \times \text{INCSTEP} \times \text{MODPERIOD}}{(2^{15} - 1) \times \text{MDF}}\right)$$

Note: The user must ensure that the product of INCSTEP and MODPERIOD is less than (2^{15} – 1).

The following values show the input setting for one possible configuration of the PLL:

- PLL input frequency: 4 MHz
- Loop divider (LDF): 64
- Input divider (IDF): 1
- VCO frequency = 4 MHz × 64 = 256 MHz
- PLL output frequency = 256 MHz/ODF
- Spread: Center spread (SPREAD_CONTROL = 0)
- Modulation frequency = 24 kHz
- Modulation depth = ±2.0% (4% pk-pk)

Using the formulae for MODPERIOD and INCSTEP:

Equation 3

MODPERIOD = Round [(4e06) / (4 × 24e03)] = Round [41.66] = 42

Equation 4

INCSTEP = Round [$((2^{15} – 1) × 2 × 64) / (100 × 5 × 42)$] = Round [199.722] = 200

Equation 5

MODPERIOD × INCSTEP = 42 × 200 = 8400 (which is less than 2^{15})

Equation 6

md(quantized)% = ((42*200*100*5) / ((2^15-1)*64) = 2.00278% (peak)

Equation 7

Error in modulation depth = 2.00278 - 2.0 = 0.00278%

If we choose MODPERIOD = 41,

Equation 8

INCSTEP = Round [$((2^{15} – 1) × 2 × 64) / (100 × 5 × 41)$] = Round [204.878] = 205

Equation 9

MODPERIOD × INCSTEP = 41 × 205 = 8405 (which is less than 2^{15})

Equation 10

md(quantized)% = $((41 × 205 × 100 × 5) / ((2^{15} – 1) × 64)$ = 2.00397% (peak)

Equation 11

Error in modulation depth = 2.00397 – 2.0 = 0.00397%

The above calculations show that the quantization error in the modulation depth depends on the flooring and rounding of MODPERIOD and INCSTEP. For this reason, the MODPERIOD and INCSTEP should be judiciously rounded/floored to minimize the quantization error in the modulation depth.

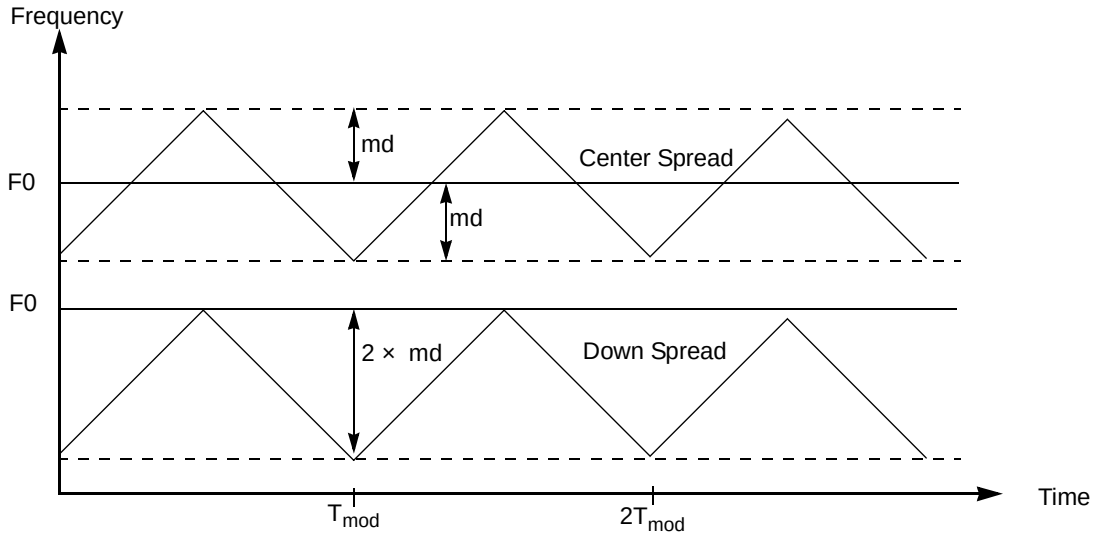


Figure 17. Frequency modulation depth spreads

Powerdown mode

To reduce consumption, the FMPLL can be switched off when not required by programming the registers ME_x_MC on the ME module.

4.9.7 Recommendations

To avoid any unpredictable behavior of the PLL clock, it is recommended to follow these guidelines:

- The PLL VCO frequency should reside in the range 256 MHz to 512 MHz. Care is required when programming the multiplication and division factors to respect this requirement.
- The user must change the multiplication, division factors only when the PLL output clock is not selected as system clock. MOD_PERIOD, INC_STEP, SPREAD_SEL bits should be modified before activating the FM modulated mode. Then strobe has to be generated to enable the new settings. If STRB_BYP is set to ‘1’ then MOD_PERIOD, INC_STEP and SPREAD_SEL can be modified only when PLL is in power down mode.
- Use progressive clock switching.

4.10 Clock Monitor Unit (CMU)

4.10.1 Overview

The Clock Monitor Unit (CMU) serves three purposes:

- Selected clock monitoring: detect if the monitored clock leaves an upper or lower frequency boundary
- XOSC clock monitoring: monitor the XOSC clock, which must be greater than the IRCOSC clock divided by a division factor given by CMU_CSR[RCDIV]
- Frequency meter: measure the frequency of the IRCOSC clock versus the reference XOSC clock frequency

When mismatch occurs in the CMU either with the PLL monitor or the XOSC monitor, the CMU notifies the RGM, ME and the FCCU (Fault Collection and Control Unit) modules. The default behavior is such that a reset occurs and a status bit is set in the RGM. The user also has the option to change the behavior of the action by disabling the reset and selecting an alternate action. The alternate action can be either entering safe mode or generating an interrupt.

Table 19. CMU module summary

Module	Monitored clocks
CMU_0	– XOSC integrity supervisor – FMPLL_0 integrity supervisor – IRCOSC frequency meter
CMU_1	System clock integrity supervisor

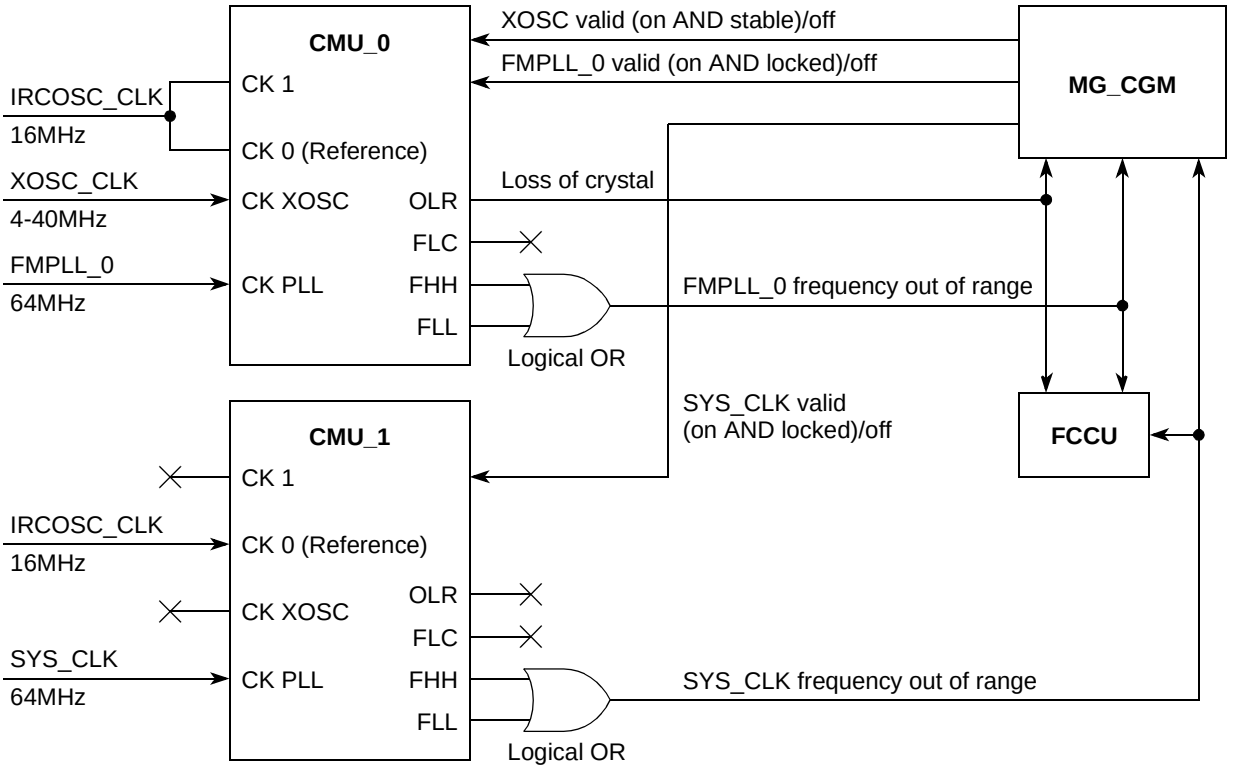


Figure 18. SPC56xP60/54 with two CMUs

4.10.2 Main features

- RC oscillator frequency measurement
- External oscillator clock monitoring with respect to CK_IRC/*n* clock
- PLL clock frequency monitoring with respect to CK_IRC/4 clock
- Event generation for various failures detected inside monitoring unit

4.10.3 Functional description

The clock and frequency names referenced in this block are defined as follows:

- CK_XOSC: clock coming from the external crystal oscillator
- CK_IRC: clock coming from the low frequency internal RC oscillator
- CK_PLL: clock coming from the PLL
- f_{XOSC}: frequency of external crystal oscillator clock
- f_{RC}: frequency of low frequency internal RC oscillator
- f_{PLL}: frequency of FMPLL clock

Crystal clock monitor[f]

If f_{XOSC} is smaller than f_{RC} divided by 2^{RCDIV} bits of CMU_0_CSR and the CK_XOSC is 'ON' and stable as signaled by the ME, then:

- An event pending bit OLRI in CMU_0_ISR is set
- A failure event OLR is signaled to the RGM and FCCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

PLL clock monitor

The PLL clock CK_PLL frequency can be monitored by programming bit CME_0 of the CMU_0_CSR to '1'. The CK_PLL monitor starts as soon as bit CME_0 is set. This monitor can be disabled at any time by writing bit CME_0 to '0'.

If the CK_PLL frequency (f_{PLL}) is greater than a reference value determined by bits HFREF[11:0] of the CMU_HFREFR and the CK_PLL is 'ON' and the PLL locked as signaled by the ME then:

- An event pending bit FHHI_0 in the CMU_0_ISR is set.
- A failure event FHH is signaled to the RGM and FCCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

If f_{PLL} is less than a reference clock frequency ($f_{RC}/4$) and the CK_PLL is 'ON' and the PLL locked as signaled by the ME, then an event pending bit FLCI_0 in the CMU_0_ISR is set.

If f_{PLL} is less than a reference value determined by bits LFREF[11:0] of the CMU_LFREFR and the CK_PLL is 'ON' and the PLL locked as signaled by the ME, then:

- An event pending bit FLLI_0 in the CMU_0_ISR is set.
- A failure event FLL is signaled to the RGM and FCCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

Note: It is possible for either the XOSC or PLL monitors to produce a false event when the XOSC or PLL frequency is too close to RC/2^{RCDIV} frequency due to an accuracy limitation of the compare circuitry.

System clock monitor

The system clock is monitored by CMU_1. The F_{SYS_CLK} frequency can be monitored by programming CMU_1_CSR[CME] = 1. SYS_CLK monitoring starts as soon as CMU_1_CSR[CME] = 1. This monitor can be disabled at any time by writing CME bit to 0.

If F_{SYS_CLK} is greater than a reference value determined by the CMU_1_HFREFR_0[HFREF] bits and the system clock is enabled, then:

- CMU_1_ISR[FHHI] is set
- A failure event is signaled to the MC_RGM and FCCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

If F_{SYS_CLK} is less than a reference clock frequency (F_{IRCOSC_CLK}÷4) and the system clock is enabled, then CMU_1_ISR[FLCI] is set.

f.Enabled only in CMU_0.

If F_{SYS_CLK} is less than a reference value determined by the CMU_1_LFREFR_0[LFREF] bits and the system clock is enabled, then:

- CMU_1_ISR[FLLI] is set
- A failure event is signaled to the MC_RGM and FCCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

Note: The system clock monitor may produce a false event when F_{SYS_CLK} is less than $2 \times F_{IRCOSC\_CLK}/2^{CMU\_1\_CSR[RCDIV]}$ due to an accuracy limitation of the compare circuitry.

Frequency meter

The frequency meter calibrates the internal RC oscillator (CK_IRC) using a known frequency. The frequency meter is implemented only on CMU_0.

Note: This value can then be stored into the flash so that application software can reuse it later on.

The reference clock will be always the XOSC. A simple frequency meter returns a draft value of CK_IRC. The measure starts when bit SFM (Start Frequency Measure) in the CMU_CSR is set to '1'. The measurement duration is given by the CMU_MDR in numbers of IRC clock cycles with a width of 20 bits. Bit SFM is reset to '0' by hardware once the frequency measurement is done and the count is loaded in the CMU_FDR. The frequency f_{RC} can be derived from the value loaded in the CMU_FDR as follows:

Equation 12

$$f_{RC} = (f_{OSC} \times MD) / n$$

where *n* is the value in the CMU_FDR and MD is the value in the CMU_MDR.

4.10.4 Memory map and register description

Table 20 shows the memory map of the CMU.

Table 20. CMU memory map

Offset from CMU_BASE (0xC3FE_0100)	Register	Location
0x0000	Control Status Register (CMU_0_CSR)	*on page 126*
0x0004	Frequency Display Register (CMU_0_FDR)	*on page 127*
0x0008	High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_0)	*on page 127*
0x000C	Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_0)	*on page 128*
0x0010	Interrupt Status Register (CMU_0_ISR)	*on page 128*
0x0014	Reserved	
0x0018	Measurement Duration Register (CMU_0_MDR)	*on page 129*
0x001C	Reserved	
0x0020	Control Status Register (CMU_1_CSR)	*on page 130*
0x0024	Reserved	

Table 20. CMU memory map (continued)

Offset from CMU_BASE (0xC3FE_0100)	Register	Location
0x0028	High Frequency Reference Register SYS_CLK (CMU_1_HFREFR_0)	*on page 130*
0x002C	Low Frequency Reference Register SYS_CLK (CMU_1_LFREFR_0)	*on page 131*
0x0030	Interrupt Status Register (CMU_1_ISR)	*on page 131*
0x0034–0x3FFF	Reserved	

Control Status Register (CMU_0_CSR)

Figure 19. Control Status Register (CMU_0_CSR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SFM	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	RCDIV[1:0]		CME_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 21. CMU_0_CSR field descriptions

Field	Description
SFM	Start frequency measure The software can only set this bit to start a clock frequency measure. It is reset by hardware when the measure is ready in the CMU_FDR. 0: Frequency measurement completed or not yet started 1: Frequency measurement not completed
RCDIV[1:0]	RC clock division factor These bits specify the RC clock division factor. The output clock is CK_IRC divided by the factor 2^{RCDIV}. This output clock is compared with CK_XOSC for crystal clock monitor feature.The clock division coding is as follows. 00: Clock divided by 1 (no division) 01: Clock divided by 2 10: Clock divided by 4 11: Clock divided by 8
CME_0	FMPLL_0 clock monitor enable 0: FMPLL_0 monitor disabled 1: FMPLL_0 monitor enabled

Frequency Display Register (CMU_0_FDR)

Figure 20. Frequency Display Register (CMU_0_FDR)

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	FD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22. CMU_0_FDR field descriptions

Field	Description
FD[19:0]	Measured frequency bits This register displays the measured frequency f_{RC} with respect to f_{OSC}. The measured value is given by the following formula: $f_{RC} = (f_{OSC} \times MD) / n$, where n is the value in CMU_FDR.

High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_0)

Figure 21. High Frequency Reference register FMPLL_0 (CMU_0_HFREFR_0)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	HFREF[11:0]											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 23. CMU_0_HFREFR_0 field descriptions

Field	Description
HFREF	High Frequency reference value These bits determine the high reference value for the FMPLL_0 clock. The reference value is given by: (HFREF[11:0]/16) × (f_{RC}/4).

Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_0)

Figure 22. Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_0)

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	LFREF[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 24. CMU_0_LFREFR_0 fields descriptions

Field	Description
LFREF	Low Frequency reference value These bits determine the low reference value for the FMPLL_0. The reference value is given by: (LFREF[11:0]/16) * (f_{RC}/4).

Interrupt Status Register (CMU_0_ISR)

Figure 23. Interrupt Status Register (CMU_0_ISR)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	FLCI_0	FHHI_0	FLLI_0	OLRI
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 25. CMU_0_ISR field descriptions

Field	Description
FLCI_0	FMPLL_0 Clock frequency less than reference clock interrupt This bit is set by hardware when CK_FMPLL_0 frequency becomes lower than reference clock frequency (f_{RC}/4) value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FLC event 1: FLC event pending

Table 25. CMU_0_ISR field descriptions (continued)

Field	Description
FHHI_0	FMPLL_0 Clock frequency higher than high reference interrupt This bit is set by hardware when CK_FMPLL_ 0 frequency becomes higher than HFREF value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FHH event 1: FHH event pending
FLLI_0	FMPLL_0 Clock frequency less than low reference event This bit is set by hardware when CK_FMPLL_0 frequency becomes lower than LFREF value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FLL event 1: FLL event pending
OLRI	Oscillator frequency less than RC frequency event This bit is set by hardware when the frequency of CK_XOSC is less than CK_IRC/2^{RCDIV} frequency and CK_XOSC is 'ON' and stable as signaled by the ME. It can be cleared by software by writing 1. 0: No OLR event 1: OLR event pending

Measurement Duration Register (CMU_0_MDR)

Figure 24. Measurement Duration Register (CMU_0_MDR)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	MD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 26. CMU_0_MDR field descriptions

Field	Description
MD[19:0]	Measurement duration bits This register displays the measured duration in term of IRC clock cycles. This value is loaded in the frequency meter downcounter. When SFM bit is set to '1', downcounter starts counting.

Control Status Register (CMU_1_CSR)

Figure 25. Control Status Register (CMU_1_CSR)

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CME_1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 27. CMU_1_CSR field descriptions

Field	Description
CME_1	SYS_CLK clock monitor enable 0 SYS_CLK monitor disabled 1 SYS_CLK monitor enabled

High Frequency Reference Register SYS_CLK (CMU_1_HFREFR_0)

Figure 26. High Frequency Reference Register SYS_CLK (CMU_1_HFREFR_0)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	HFREF[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 28. CMU_1_HFREFR_0 field descriptions

Field	Description
HFREF[11:0]	High Frequency reference value These bits determine the high reference value for the SYS_CLK clock. The reference value is given by: (HFREF[11:0]/16) × (F_{RC}/4).

Low Frequency Reference Register SYS_CLK (CMU_1_LFREFR_0)

Figure 27. Low Frequency Reference Register SYS_CLK (CMU_1_LFREFR_0)

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	LFREF[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 29. CMU_1_LFREFR_0 field descriptions

Field	Description
LFREF[11:0]	Low Frequency reference value These bits determine the low reference value for the SYS_CLK clock. The reference value is given by: (LFREF[11:0]/16) × (f_{RC}/4).

Interrupt Status Register (CMU_1_ISR)

Figure 28. Interrupt Status register (CMU_1_ISR)

Address Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	FLCI_1	FHHI_1	FLLI_1	0
W													w1c	w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 30. CMU_1_ISR field descriptions

Field	Description
FLCI_1	SYS_CLK clock frequency less than reference clock event. This bit is set by hardware when the SYS_CLK clock frequency is lower than the reference clock frequency (FRC/4) value and SYS_CLK is ‘ON’ and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0 No FLC event. 1 FLC event is pending.

Table 30. CMU_1_ISR field descriptions

FHHI_1	SYS_CLK clock frequency higher than high reference event. This bit is set by hardware when the SYS_CLK frequency is higher than the HFREF value and SYS_CLK is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0 No FHH event. 1 FHH event is pending.
FLLI_1	SYS_CLK clock frequency less than low reference event. This bit is set by hardware when the SYS_CLK frequency becomes lower than LFREF value and SYS_CLK is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0 No FLL event. 1 FLL event is pending.

5 Clock Generation Module (MC_CGM)

5.1 Introduction

5.1.1 Overview

The clock generation module (MC_CGM) generates reference clocks for all the chip blocks. The MC_CGM selects one of the system clock sources to supply the system clock. The MC_ME controls the system clock selection (see the MC_ME chapter for more details). A set of MC_CGM registers controls the clock dividers which are used for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources which have addressable memory spaces are accessed through the MC_CGM memory space. The MC_CGM also selects and generates an output clock.

Figure 29 shows the MC_CGM block diagram.

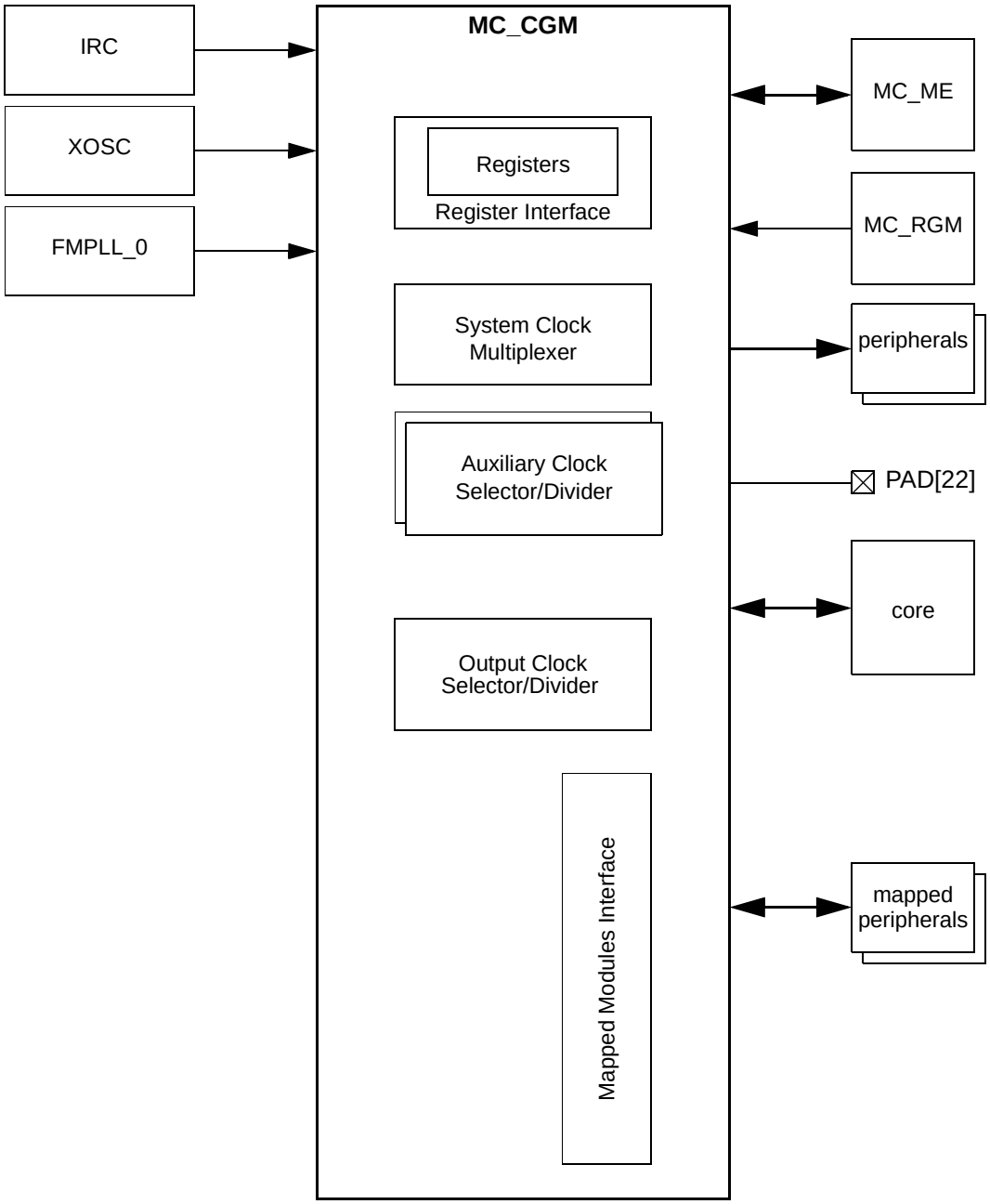


Figure 29. MC_CGMBlock Diagram

5.1.2 Features

The MC_CGM includes the following features:

- generates system and peripheral clocks
- selects and enables/disables the system clock supply from system clock sources according to MC_ME control
- contains a set of registers to control clock dividers for divided clock generation
- supports multiple clock sources and maps their address spaces to its memory map
- generates an output clock
- guarantees glitch-less clock transitions when changing the system clock selection
- supports 8, 16, and 32-bit wide read/write accesses

5.2 External Signal Description

The MC_CGM delivers an output clock to the PAD[22] pin for off-chip use and/or observation.

5.3 Memory Map and Register Definition

Table 31. MC_CGM Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_0370	CGM_OC_EN	Output Clock Enable	word	read	read/write	read/write	*on page 139*
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read	read/write	read/write	*on page 140*
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	read	read	*on page 141*
0xC3FE_0380	CGM_AC0_SC	Aux Clock 0 Select Control	word	read	read/write	read/write	*on page 142*
0xC3FE_0384	CGM_AC0_DC0	Aux Clock 0 Divider Configuration 0	byte	read	read/write	read/write	*on page 142*
0xC3FE_0388	CGM_AC1_SC	Aux Clock 1 Select Control	word	read	read/write	read/write	*on page 143*
0xC3FE_038C	CGM_AC1_DC0	Aux Clock 1 Divider Configuration 0	byte	read	read/write	read/write	*on page 144*
0xC3FE_0390	CGM_AC2_SC	Aux Clock 2 Select Control	word	read	read/write	read/write	*on page 145*
0xC3FE_0394	CGM_AC2_DC0	Aux Clock 2 Divider Configuration 0	byte	read	read/write	read/write	*on page 145*

Table 31. MC_CGM Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_0398	CGM_AC3_SC	Aux Clock 3 Select Control	word	read	read/write	read/write	*on page 146*
0xC3FE_039C	CGM_AC3_DC0	Aux Clock 3 Divider Configuration 0	byte	read	read/write	read/write	*on page 147*

Note: Any access to unused registers as well as write accesses to read-only registers will:

- *not change register content*
- *cause a transfer error*

Table 32. MC_CGM Memory Map

Address	Name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0000 … 0xC3FE_001C	XOSC registers																
0xC3FE_0020 … 0xC3FE_005F	reserved																
0xC3FE_0060 … 0xC3FE_007C	IRC registers																
0xC3FE_0080 … 0xC3FE_009F	reserved																
0xC3FE_00A0 … 0xC3FE_00BC	FMPLL_0 registers																
0xC3FE_00C0 … 0xC3FE_00FF	reserved																

Table 32. MC_CGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0100 … 0xC3FE_011C	CMU0 registers																	
0xC3FE_0120 … 0xC3FE_013C	CMU1 registers																	
0xC3FE_0140 … 0xC3FE_015C	reserved																	
0xC3FE_0160 … 0xC3FE_017C	CLK_DIV_256 registers																	
0xC3FE_0180 … 0xC3FE_036F	reserved																	
0xC3FE_0370	CGM_OC_EN	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
		W																
0xC3FE_0374	CGM_OCDS_SC	R	0	0	SELDIV		SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0378	CGM_SC_SS	R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_037C	reserved																	

Table 32. MC_CGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0380	CGM_AC0_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0384	CGM_AC0_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0388	CGM_AC1_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_038C	CGM_AC1_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0390	CGM_AC2_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0394	CGM_AC2_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0398	CGM_AC3_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 32. MC_CGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_039C	CGM_AC3_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0400 … 0xC3FE_3FFC	reserved																	

5.3.1 Register Descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the **CGM_OC_EN** register may be accessed as a word at address 0xC3FE_0370, as a half-word at address 0xC3FE_0372, or as a byte at address 0xC3FE_0373.

Output Clock Enable Register (CGM_OC_EN)

Figure 30. Output Clock Enable Register (CGM_OC_EN)

Address 0xC3FE_0370 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to enable and disable the output clock.

Table 33. Output Clock Enable Register (CGM_OC_EN) Field Descriptions

Field	Description
EN	**Output Clock Enable control** 0 Output Clock is disabled 1 Output Clock is enabled

Output Clock Division Select Register (CGM_OCDS_SC)

Figure 31. Output Clock Division Select Register (CGM_OCDS_SC)

Address 0xC3FE_0374 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	0	0	SELDIV		SELCTL			
W								
Reset	0	0	0	0	0	0	0	0

This register is used to select the current output clock source and by which factor it is divided before being delivered at the output clock.

Table 34. Output Clock Enable Register (CGM_OC_EN) Field Descriptions

Field	Description
SELDIV	**Output Clock Division Select** 00 output selected Output Clock without division 01 output selected Output Clock divided by 2 10 output selected Output Clock divided by 4 11 output selected Output Clock divided by 8
SELCTL	**Output Clock Source Selection Control** — This value selects the current source for the output clock. 0000 IRC 0001 XOSC 0010 FMPLL_0 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

System Clock Select Status Register (CGM_SC_SS)

Figure 32. System Clock Select Status Register (CGM_SC_SS)

Address 0xC3FE_0378 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the current system clock source selection.

Table 35. System Clock Select Status Register (CGM_SC_SS) Field Descriptions

Field	Description
SELSTAT	**System Clock Source Selection Status** — This value indicates the current source for the system clock. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 PCS 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled

Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

Figure 33. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

Address 0xC3FE_0380 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to select the current clock source for the following clocks:

- undivided: (unused)
- divided by auxiliary clock 0 divider 0: (unused)

See *Figure 42* for details.

Table 36. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 0 Source Selection Control** — This value selects the current source for auxiliary clock 0. 0000 (no clock) 0001 reserved 0010 (no clock) 0011 reserved 0100 (no clock) 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

Auxiliary Clock 0 Divider 0 Configuration Register (CGM_AC0_DC0)

Figure 34. Auxiliary Clock 0 Divider 0 Configuration Register (CGM_AC0_DC0)

Address0xC3FE_0384 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

This register controls auxiliary clock 0 divider 0.

Table 37. Auxiliary Clock 0 Divider 0 Configuration Register (CGM_AC0_DC0) Field Descriptions

Field	Description
DE0	**Divider 0 Enable** 0 Disable auxiliary clock 0 divider 0 1 Enable auxiliary clock 0 divider 0
DIV0	**Divider 0 Division Value** — The resultant (unused) will have a period 'DIV0 + 1' times that of auxiliary clock 0. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the (unused) remains disabled.

Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

Figure 35. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

Address 0xC3FE_0388 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to select the current clock source for the following clocks:

- undivided: (unused)
- divided by auxiliary clock 1 divider 0: (unused)

Table 38. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 1 Source Selection Control** — This value selects the current source for auxiliary clock 1. 0000 (no clock) 0001 reserved 0010 reserved 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0)

Figure 36. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0)

Address0xC3FE_038C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

These registers controls auxiliary clock 1 divider 0.

Table 39. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0) Field Descriptions

Field	Description
DE0	**Divider 0 Enable** 0 Disable auxiliary clock 1 divider 0 1 Enable auxiliary clock 1 divider 0
DIV0	**Divider 0 Division Value** — The resultant (unused) will have a period 'DIV0 + 1' times that of auxiliary clock 1. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the (unused) remains disabled.

Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

Figure 37. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

Address 0xC3FE_0390 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to select the current clock source for the following clock:s

- undivided: (unused)
- divided by auxiliary clock 2 divider 0: (unused)

See *Figure 44* for details.

Table 40. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 2 Source Selection Control** — This value selects the current source for auxiliary clock 2. 0000 (no clock) 0001 reserved 0010 (no clock) 0011 reserved 0100 (no clock) 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

Auxiliary Clock 2 Divider 0 Configuration Register (CGM_AC2_DC0)

Figure 38. Auxiliary Clock 2 Divider 0 Configuration Register (CGM_AC2_DC0)

Address0xC3FE_0394 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

These registers controls auxiliary clock 2 divider 0.

Table 41. Auxiliary Clock 2 Divider 0 Configuration Register (CGM_AC2_DC0) Field Descriptions

Field	Description
DE0	**Divider 0 Enable** 0 Disable auxiliary clock 2 divider 0 1 Enable auxiliary clock 2 divider 0
DIV0	**Divider 0 Division Value** — The resultant (unused) will have a period 'DIV0 + 1' times that of auxiliary clock 2. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the (unused) remains disabled.

Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)

Figure 39. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)

Address 0xC3FE_0398 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is used to select the current clock source for the following clocks:

- undivided: (unused)
- divided by auxiliary clock 3 divider 0: FlexRay clock

See *Figure 45* for details.

Table 42. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 3 Source Selection Control** — This value selects the current source for auxiliary clock 3. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 0101 reserved 0110 reserved 0111 reserved 1000 FMPLL_0_D1 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

Auxiliary Clock 3 Divider 0 Configuration Register (CGM_AC3_DC0)

Figure 40. Auxiliary Clock 3 Divider 0 Configuration Register (CGM_AC3_DC0)

Address0xC3FE_039C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

These registers controls auxiliary clock 3 dividers 0.

Table 43. Auxiliary Clock 3 Divider 0 Configuration Register (CGM_AC3_DC0) Field Descriptions

Field	Description
DE0	**Divider 0 Enable** 0 Disable auxiliary clock 3 divider 0 1 Enable auxiliary clock 3 divider 0
DIV0	**Divider 0 Division Value** — The resultant FlexRay clock will have a period 'DIV0 + 1' times that of auxiliary clock 3. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the FlexRay clock remains disabled.

5.4 Functional Description

5.4.1 System Clock Generation

Figure 41 shows the block diagram of the system clock generation logic. The MC_ME provides the system clock select and switch mask (see MC_ME chapter for more details),

and the MC_RGM provides the safe clock request (see MC_RGM chapter for more details). The safe clock request forces the selector to select the IRC as the system clock and to ignore the system clock select.

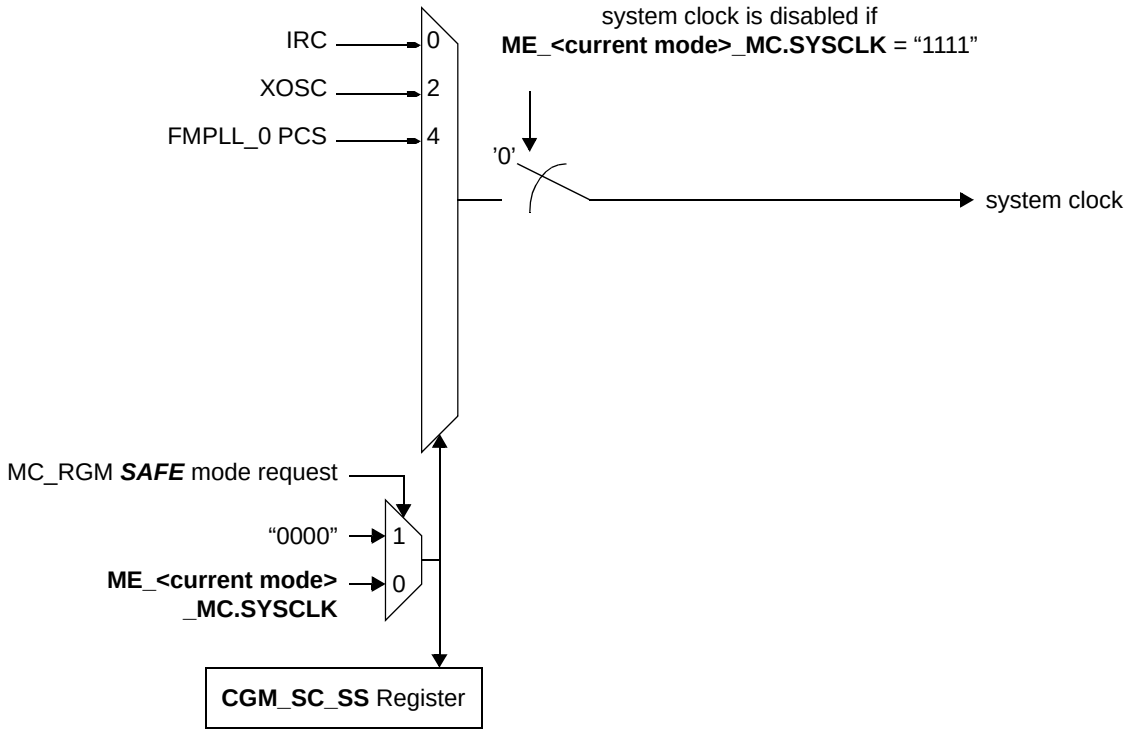


Figure 41. MC_CGM System Clock Generation Overview

System Clock Source Selection

During normal operation, the system clock selection is controlled

- on a ***SAFE*** mode or reset event, by the MC_RGM
- otherwise, by the MC_ME

System Clock Disable

During the ***TEST*** mode, the system clock can be disabled by the MC_ME.

5.4.2 Auxiliary Clock Generation

Figure 41 shows the block diagram of the auxiliary clock generation logic. See *Section : Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)*, *Section : Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)*, *Section : Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)*, and *Section : Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)* for auxiliary clock selection control.

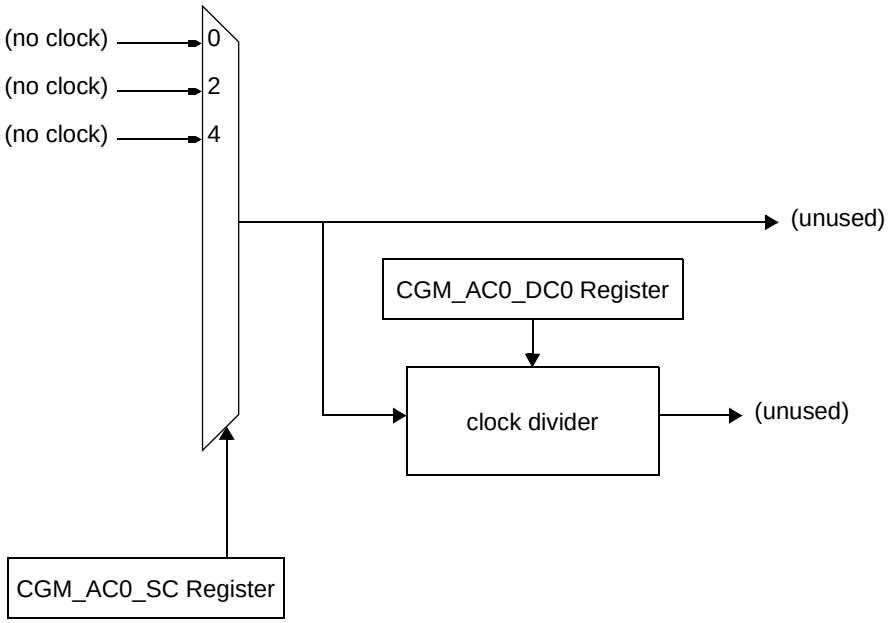


Figure 42. MC_CGM Auxiliary Clock 0 Generation Overview

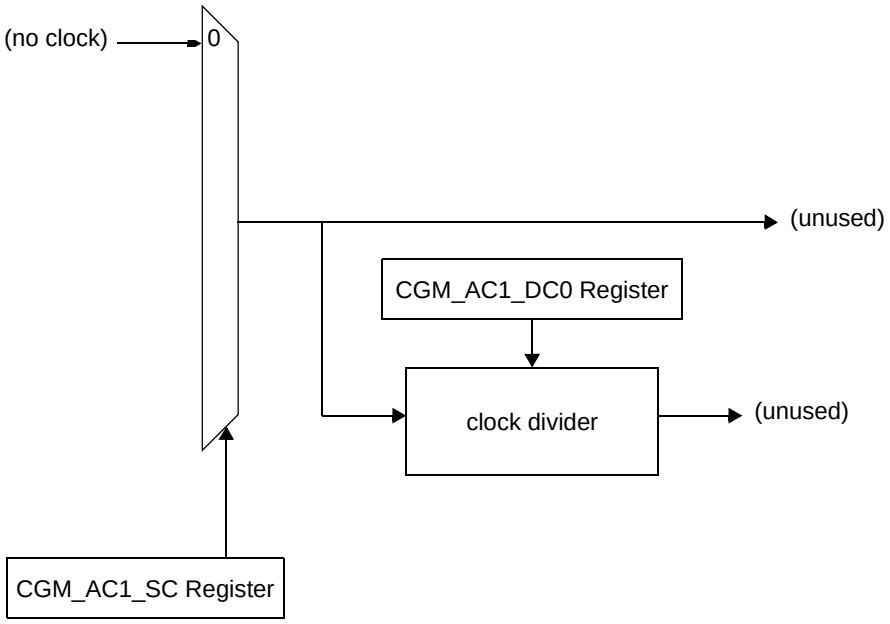


Figure 43. MC_CGM Auxiliary Clock 1 Generation Overview

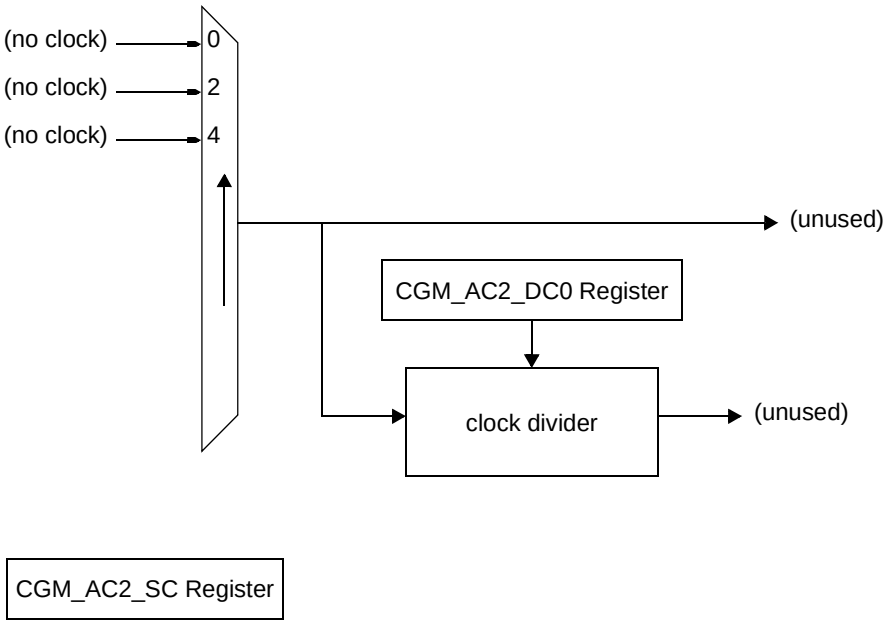


Figure 44. MC_CGM Auxiliary Clock 2 Generation Overview

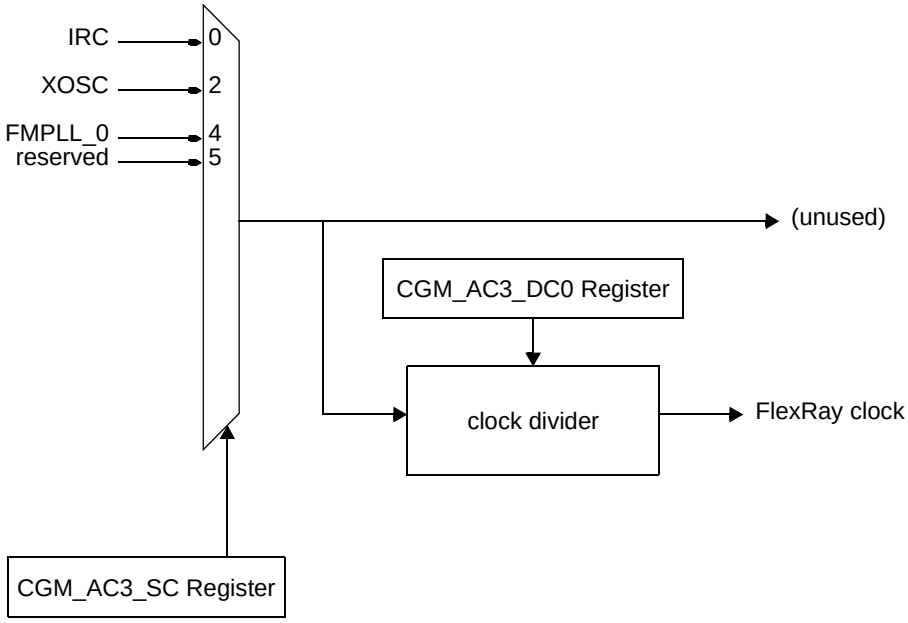


Figure 45. MC_CGM Auxiliary Clock 3 Generation Overview

Auxiliary Clock Dividers

The MC_CGM generates the following derived clocks:

- (unused) - controlled by the **CGM_AC0_DC0** register
- (unused) - controlled by the **CGM_AC1_DC0** register
- (unused) - controlled by the **CGM_AC2_DC0** register
- FlexRay clock - controlled by the **CGM_AC3_DC0** register

5.4.3 Dividers Functional Description

Dividers are used for the generation of divided system and peripheral clocks. The MC_CGM has the following control registers for built-in dividers:

- *Section : Auxiliary Clock 0 Divider 0 Configuration Register (CGM_AC0_DC0)*
- *Section : Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0)*
- *Section : Auxiliary Clock 2 Divider 0 Configuration Register (CGM_AC2_DC0)*
- *Section : Auxiliary Clock 3 Divider 0 Configuration Register (CGM_AC3_DC0)*

The reset value of all counters is ‘1’. If a divider has its **DE** bit in the respective configuration register set to ‘0’ (the divider is disabled), any value in its **DIV*n*** field is ignored.

5.4.4 Output Clock Multiplexing

The MC_CGM contains a multiplexing function for a number of clock sources which can then be used as output clock sources. The selection is done via the **CGM_OCDS_SC** register.

5.4.5 Output Clock Division Selection

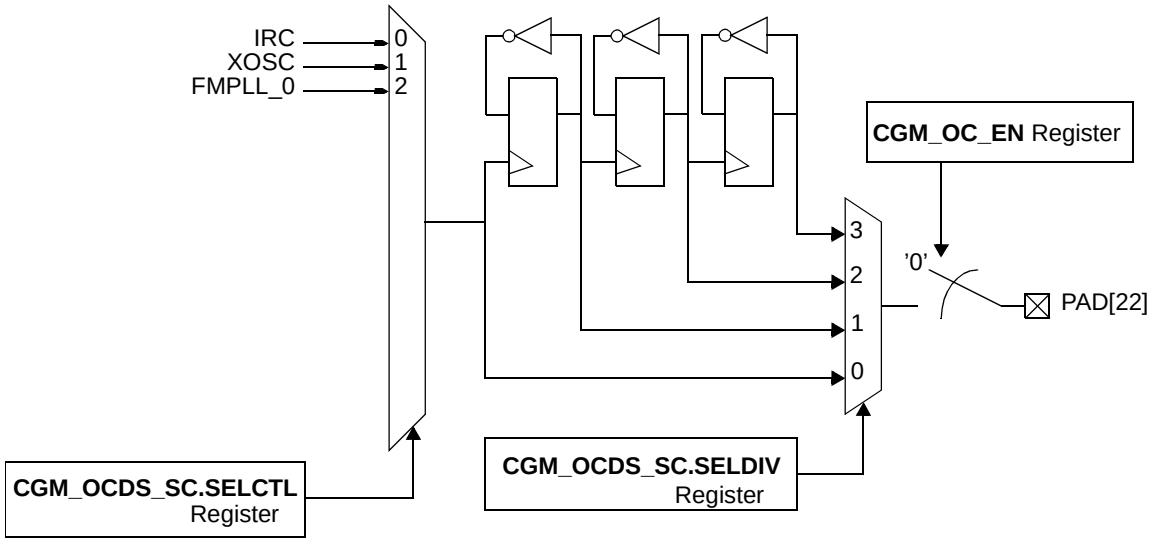


Figure 46. MC_CGM Output Clock Multiplexer and PAD[22] Generation

The MC_CGM provides the following output signal for the output clock generation:

- PAD[22] (see *Figure 46*). This signal is generated by using one of the 3-stage ripple counter outputs or the selected signal without division. The non-divided signal is not

guaranteed to be 50% duty cycle by the MC_CGM.

The MC_CGM also has an output clock enable register (see *Section : Output Clock Enable Register (CGM_OC_EN)*) that contains the output clock enable/disable control bit.

6 Mode Entry Module (MC_ME)

6.1 Introduction

6.1.1 Overview

The MC_ME controls the chip mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application.

Figure 47 shows the MC_ME block diagram.

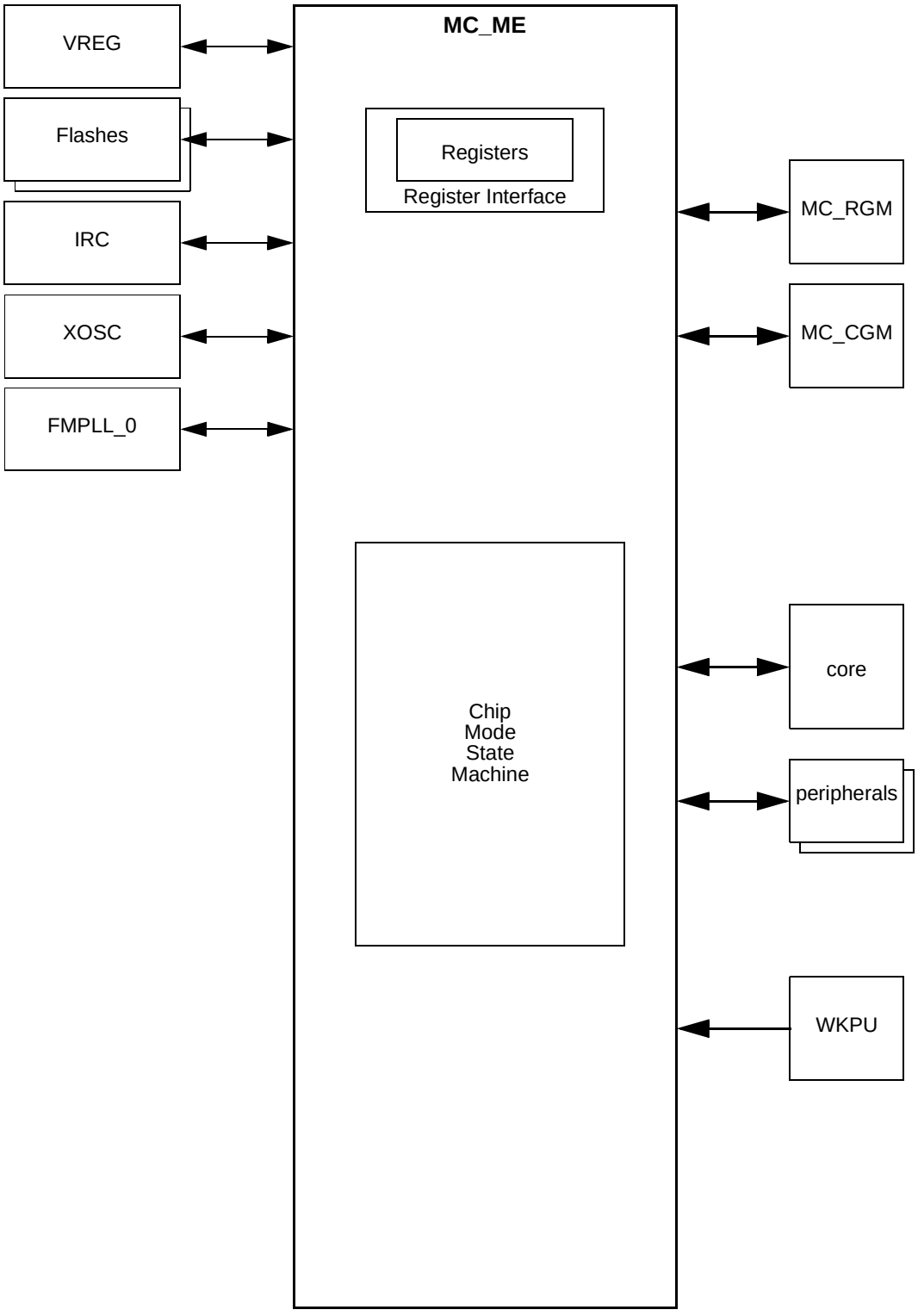


Figure 47. MC_ME Block Diagram

6.1.2 Features

The MC_ME includes the following features:

- control of the available modes by the **ME_ME** register
- definition of various chip mode configurations by the **ME_<*mode*>_MC** registers
- control of the actual chip mode by the **ME_MCTL** register
- capture of the current mode and various resource status within the contents of the **ME_GS** register
- optional generation of various mode transition interrupts
- status bits for each cause of invalid mode transitions
- peripheral clock gating control based on the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL*n*** registers
- capture of current peripheral clock gated/enabled status

6.1.3 Modes of Operation

The MC_ME is based on several chip modes corresponding to different usage models of the chip. Each mode is configurable and can define a policy for energy and processing power management to fit particular system requirements. An application can easily switch from one mode to another depending on the current needs of the system. The operating modes controlled by the MC_ME are divided into system and user modes. The system modes are modes such as ***RESET***, ***DRUN***, ***SAFE***, and ***TEST***. These modes aim to ease the configuration and monitoring of the system. The user modes are modes such as ***RUN0…3***, ***HALT0***, and ***STOP0*** which can be configured to meet the application requirements in terms of energy management and available processing power. The modes ***DRUN***, ***SAFE***, ***TEST***, and ***RUN0…3*** are the chip software running modes.

Table 44 describes the MC_ME modes.

Table 44. MC_ME Mode Descriptions

Name	Description	Entry	Exit
RESET	This is a chip-wide virtual mode during which the application is not active. The system remains in this mode until all resources are available for the embedded software to take control of the chip. It manages hardware initialization of chip configuration, voltage regulators, clock sources, and flash modules.	system reset assertion from MC_RGM	system reset deassertion from MC_RGM
DRUN	This is the entry mode for the embedded software. It provides full accessibility to the system and enables the configuration of the system at startup. It provides the unique gate to enter user modes. BAM when present is executed in ***DRUN*** mode.	system reset deassertion from MC_RGM, software request from ***SAFE***, ***TEST*** and ***RUN0…3***	system reset assertion, ***RUN0…3***, ***TEST*** via software, ***SAFE*** via software or hardware failure.
SAFE	This is a chip-wide service mode which may be entered on the detection of a recoverable error. It forces the system into a pre-defined safe configuration from which the system may try to recover.	hardware failure, software request from ***DRUN***, ***TEST***, and ***RUN0…3***	system reset assertion, ***DRUN*** via software

Table 44. MC_ME Mode Descriptions (continued)

Name	Description	Entry	Exit
TEST	This is a chip-wide service mode which is intended to provide a control environment for chip software testing.	software request from ***DRUN***	system reset assertion, ***DRUN*** via software
RUN0...3	These are software running modes where most processing activity is done. These various run modes allow to enable different clock & power configurations of the system with respect to each other.	software request from ***DRUN*** or other ***RUN0...3***, interrupt event from ***HALT0,*** interrupt or wakeup event from ***STOP0***	system reset assertion, ***SAFE*** via software or hardware failure, other ***RUN0...3*** modes, ***HALT0***, ***STOP0*** via software
HALT0	This is a reduced-activity low-power mode during which the clock to the core is disabled. It can be configured to switch off analog peripherals like clock sources, flash, main regulator, etc. for efficient power management at the cost of higher wakeup latency.	software request from ***RUN0...3***	system reset assertion, ***SAFE*** on hardware failure, ***RUN0...3*** on interrupt event
STOP0	This is an advanced low-power mode during which the clock to the core is disabled. It may be configured to switch off most of the peripherals including clock sources for efficient power management at the cost of higher wakeup latency.	software request from ***RUN0...3***	system reset assertion, ***SAFE*** on hardware failure, ***RUN0...3*** on interrupt event or wakeup event

6.2 External Signal Description

The MC_ME has no connections to any external pins.

6.3 Memory Map and Register Definition

The MC_ME contains registers for:

- mode selection and status reporting
- mode configuration
- mode transition interrupts status and mask control
- scalable number of peripheral sub-mode selection and status reporting

6.3.1 Memory Map

Table 45. MC_ME Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C000	ME_GS	Global Status	word	read	read	read	*on page 165*
0xC3FD_C004	ME_MCTL	Mode Control	word	read	read/write	read/write	*on page 167*

Table 45. MC_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C008	ME_ME	Mode Enable	word	read	read/write	read/write	on page 168
0xC3FD_C00C	ME_IS	Interrupt Status	word	read	read/write	read/write	on page 170
0xC3FD_C010	ME_IM	Interrupt Mask	word	read	read/write	read/write	on page 171
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read	read/write	read/write	on page 172
0xC3FD_C018	ME_DMTS	Debug Mode Transition Status	word	read	read	read	on page 173
0xC3FD_C020	ME_RESET_MC	***RESET*** Mode Configuration	word	read	read	read	on page 176
0xC3FD_C024	ME_TEST_MC	***TEST*** Mode Configuration	word	read	read/write	read/write	on page 177
0xC3FD_C028	ME_SAFE_MC	***SAFE*** Mode Configuration	word	read	read/write	read/write	on page 177
0xC3FD_C02C	ME_DRUN_MC	***DRUN*** Mode Configuration	word	read	read/write	read/write	on page 178
0xC3FD_C030	ME_RUN0_MC	***RUN0*** Mode Configuration	word	read	read/write	read/write	on page 178
0xC3FD_C034	ME_RUN1_MC	***RUN1*** Mode Configuration	word	read	read/write	read/write	on page 178
0xC3FD_C038	ME_RUN2_MC	***RUN2*** Mode Configuration	word	read	read/write	read/write	on page 178
0xC3FD_C03C	ME_RUN3_MC	***RUN3*** Mode Configuration	word	read	read/write	read/write	on page 178
0xC3FD_C040	ME_HALT0_MC	***HALT0*** Mode Configuration	word	read	read/write	read/write	on page 179
0xC3FD_C048	ME_STOP0_MC	***STOP0*** Mode Configuration	word	read	read/write	read/write	on page 179
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	read	read	on page 181
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	read	read	on page 182
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	read	read	on page 182
0xC3FD_C06C	ME_PS3	Peripheral Status 3	word	read	read	read	on page 183
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read	read/write	read/write	on page 183

Table 45. MC_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read	read/write	read/write	on page 183
0xC3FD_C088	ME_RUN_PC2	Run Peripheral Configuration 2	word	read	read/write	read/write	on page 183
0xC3FD_C08C	ME_RUN_PC3	Run Peripheral Configuration 3	word	read	read/write	read/write	on page 183
0xC3FD_C090	ME_RUN_PC4	Run Peripheral Configuration 4	word	read	read/write	read/write	on page 183
0xC3FD_C094	ME_RUN_PC5	Run Peripheral Configuration 5	word	read	read/write	read/write	on page 183
0xC3FD_C048	ME_RUN_PC6	Run Peripheral Configuration 6	word	read	read/write	read/write	on page 183
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read	read/write	read/write	on page 183
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read	read/write	read/write	on page 184
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read	read/write	read/write	on page 184
0xC3FD_C0A8	ME_LP_PC2	Low-Power Peripheral Configuration 2	word	read	read/write	read/write	on page 184
0xC3FD_C0AC	ME_LP_PC3	Low-Power Peripheral Configuration 3	word	read	read/write	read/write	on page 184
0xC3FD_C0B0	ME_LP_PC4	Low-Power Peripheral Configuration 4	word	read	read/write	read/write	on page 184
0xC3FD_C0B4	ME_LP_PC5	Low-Power Peripheral Configuration 5	word	read	read/write	read/write	on page 184
0xC3FD_C0B8	ME_LP_PC6	Low-Power Peripheral Configuration 6	word	read	read/write	read/write	on page 184
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read	read/write	read/write	on page 184
0xC3FD_C0C4	ME_PCTL4	DSPI0 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0C5	ME_PCTL5	DSPI1 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0C6	ME_PCTL6	DSPI2 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0C7	ME_PCTL7	DSPI3 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0C8	ME_PCTL8	DSPI4 Control	byte	read	read/write	read/write	on page 185

Table 45. MC_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C0D0	ME_PCTL16	FlexCAN0 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0D1	ME_PCTL17	FlexCAN1 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0D8	ME_PCTL24	FlexRay Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0DA	ME_PCTL26	SafetyPort Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0E0	ME_PCTL32	ADC0 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0E3	ME_PCTL35	CTU0 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0E6	ME_PCTL38	eTimer0 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0E7	ME_PCTL39	eTimer1 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0F0	ME_PCTL48	LIN_FLEX0 Control	byte	read	read/write	read/write	on page 185
0xC3FD_C0F1	ME_PCTL49	LIN_FLEX1 Control	byte	read	read/write	read/write	on page 185
...							
0xC3FD_C11C	ME_PCTL92	PIT_RTI Control	byte	read	read/write	read/write	on page 185
0xC3FD_C13F	ME_PCTL127	CORE1 Control	byte	read	read/write	read/write	on page 185

Note: Any access to unused registers as well as write accesses to read-only registers will:

- *not change register content*
- *cause a transfer error*

Table 46. MC_ME Memory Map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	1	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
		W																
		R	0	0	0	0	0	0	0	0	0	S_FMPLL_0	S_XOSC	S_IRC	S_SYSCLK			
		W																
0xC3FD_C004	ME_MCTL	R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
		W	KEY															
0xC3FD_C008	ME_ME	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET_FUNC
		W																
0xC3FD_C00C	ME_IS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	I_ICONF_CU	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C010	ME_IM	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	M_ICONF_CU	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																

Table 46. MC_ME Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C014	ME_IMTS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C018	ME_DMTS	R	PREVIOUS_MODE				0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
		W																
		R	0	VREG_CSRC_SC	CSRC_CSRC_SC	IRC_SC	SCSRC_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	CDP_PRPH_96_127	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
		W																
0xC3FD_C01C	reserved																	
0xC3FD_C020	ME_RESET_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																
0xC3FD_C024	ME_TEST_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																

Table 46. MC_ME Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																
0xC3FD_C02C	ME_DRUN_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																
0xC3FD_C030 … 0xC3FD_C03C	ME_RUN0…3_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																
0xC3FD_C040	ME_HALT0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																
0xC3FD_C044	reserved																	

Table 46. MC_ME Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C048	ME_STOP0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
		W																
0xC3FD_C04C ... 0xC3FD_C05C	reserved																	
0xC3FD_C060	ME_PS0	R	0	0	0	0	0	S_SafetyPort	0	S_FlexRay	0	0	0	0	0	0	S_FlexCAN1	S_FlexCAN0
		W																
		R	0	0	0	0	0	0	0	S_DSPI4	S_DSPI3	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	0	0	S_CRC1	0	S_CRC0	0	0	0	0	0	0	0	0	S_LIN_FLEX1	S_LIN_FLEX0
		W																
		R	0	0	0	0	0	0	0	0	S_eTimer1	S_eTimer0	0	0	S_CTU0	0	0	S_ADC0
		W																
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT_RTI	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 46. MC_ME Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C06C	ME_PS3	R	S_CORE1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FD_C070	reserved																	
0xC3FD_C074 … 0xC3FD_C07C	reserved																	
0xC3FD_C080 … 0xC3FD_C09C	ME_RUN_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		W																
0xC3FD_C0A0 … 0xC3FD_C0BC	ME_LP_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	STOP0	0	HALT0	0	0	0	0	0	0	0	0
		W																
0xC3FD_C0C0 … 0xC3FD_C14C	ME_PCTL0…143[1]	R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG		
		W																
		R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG		
		W																
0xC3FD_C150 … 0xC3FD_FFFC	reserved																	

1. There is space in the register map for 144 peripherals. Please refer to *Table 45* for the **ME_PCTL*n*** locations actually occupied. The unoccupied locations contain a read-only byte value of 0x00.

6.3.2 Register Description

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the **ME_RUN_PC0** register may be accessed as a word at address 0xC3FD_C080, as a half-word at address 0xC3FD_C082, or as a byte at address 0xC3FD_C083.

Some fields may be read-only, and their reset value of ‘1’ or ‘0’ and the corresponding behavior cannot be changed.

Global Status Register (ME_GS)

Figure 48. Global Status Register (ME_GS)

Address 0xC3FD_C000 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	S_CURRENT_MODE				S_MTRANS	1	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	S_FMPLL_0	S_XOSC	S_IRC	S_SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register contains global mode status.

Table 47. Global Status Register (ME_GS) Field Descriptions

Field	Description
S_CURRENT_MODE	**Current chip mode status** 0000 ***RESET*** 0001 ***TEST*** 0010 ***SAFE*** 0011 ***DRUN*** 0100 ***RUN0*** 0101 ***RUN1*** 0110 ***RUN2*** 0111 ***RUN3*** 1000 ***HALT0*** 1001 reserved 1010 ***STOP0*** 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved
S_MTRANS	**Mode transition status** 0 Mode transition process is not active 1 Mode transition is ongoing

Table 47. Global Status Register (ME_GS) Field Descriptions (continued)

Field	Description
S_PDO	**Output power-down status** — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In ***SAFE***/***TEST*** modes, outputs of pads are forced to high impedance state and the pads power sequence driver is disabled. The inputs are level unchanged. In ***STOP0*** mode, only the pad power sequence driver is disabled, but the state of the output remains functional.
S_MVR	**Main voltage regulator status** 0 Main voltage regulator is not ready 1 Main voltage regulator is ready for use
S_DFLA	**Data flash availability status** 00 Data flash is not available 01 Data flash is in power-down mode 10 Data flash is not available 11 Data flash is in normal mode and available for use
S_CFLA	Code flash **availability status** 00 Code flash is not available 01 Code flash is in power-down mode 10 Code flash is in low-power mode 11 Code flash is in normal mode and available for use
S_FMPLL_0	**system PLL status** 0 system PLL is not stable 1 system PLL is providing a stable clock
S_XOSC	**external oscillator status** 0 external oscillator is not stable 1 external oscillator is providing a stable clock
S_IRC	**internal RC oscillator status** 0 internal RC oscillator is not stable 1 internal RC oscillator is providing a stable clock
S_SYSCLK	**System clock switch status** — These bits specify the system clock currently used by the system. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 PCS 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled

Mode Control Register (ME_MCTL)

Figure 49. Mode Control Register (ME_MCTL)

Address 0xC3FD_C004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
W	KEY															
Reset	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1

This register is used to trigger software-controlled mode changes. Depending on the modes as enabled by **ME_ME** register bits, configurations corresponding to unavailable modes are reserved and access to **ME_<*mode*>_MC** registers must respect this for successful mode requests.

Note: Byte and half-word write accesses are not allowed for this register as a predefined key is required to change its value.

Table 48. Mode Control Register (ME_MCTL) Field Descriptions

Field	Description
TARGET_MODE	T**arget chip mode** — These bits provide the target chip mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. These bits are automatically updated by hardware while entering ***SAFE*** on hardware request. Also, while exiting from the ***HALT0*** and ***STOP0*** modes on hardware exit events, these are updated with the appropriate ***RUN0...3*** mode value. 0000 ***RESET*** (triggers a 'functional' reset event) 0001 ***TEST*** 0010 ***SAFE*** 0011 ***DRUN*** 0100 ***RUN0*** 0101 ***RUN1*** 0110 ***RUN2*** 0111 ***RUN3*** 1000 ***HALT0*** 1001 reserved 1010 ***STOP0*** 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved
KEY	**Control key** — These bits enable write access to this register. Any write access to the register with a value different from the keys is ignored. Read access will always return inverted key. KEY:0101101011110000 (0x5AF0) INVERTED KEY:1010010100001111 (0xA50F)

Mode Enable Register (ME_ME)

Figure 50. Mode Enable Register (ME_ME)

Address 0xC3FD_C008 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET_FUNC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	01

This register allows a way to disable the chip modes which are not required for a given chip. ***RESET***, ***SAFE***, ***DRUN***, and ***RUN0*** modes are always enabled.

Table 49. Mode Enable Register (ME_ME) Field Descriptions

Field	Description
STOP0	***STOP0*** **mode enable** 0 ***STOP0*** mode is disabled 1 ***STOP0*** mode is enabled
HALT0	***HALT0*** **mode enable** 0 ***HALT0*** mode is disabled 1 ***HALT0*** mode is enabled
RUN3	***RUN3*** **mode enable** 0 ***RUN3*** mode is disabled 1 ***RUN3*** mode is enabled
RUN2	***RUN2*** **mode enable** 0 ***RUN2*** mode is disabled 1 ***RUN2*** mode is enabled
RUN1	***RUN1*** **mode enable** 0 ***RUN1*** mode is disabled 1 ***RUN1*** mode is enabled
RUN0	***RUN0*** **mode enable** 1 ***RUN0*** mode is enabled
DRUN	***DRUN*** **mode enable** 1 ***DRUN*** mode is enabled
SAFE	***SAFE*** **mode enable** 1 ***SAFE*** mode is enabled
TEST	***TEST*** **mode enable** 0 ***TEST*** mode is disabled 1 ***TEST*** mode is enabled
RESET_FUNC	**'functional'** ***RESET*** **mode enable** 1 **'functional'** ***RESET*** mode is enabled

Interrupt Status Register (ME_IS)

Figure 51. Interrupt Status Register (ME_IS)

Address 0xC3FD_C00C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	I_ICONF_CU	I_ICONF	I_IMODE	I_SAFE	I_MTC
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the current interrupt status.

Table 50. Interrupt Status Register (ME_IS) Field Descriptions

Field	Description
I_ICONF_CU	**Invalid mode configuration interrupt (Clock Usage)** — This bit is set during a mode transition if a clock which is required to be on by an enabled peripheral is configured to be turned off. It is cleared by writing a ‘1’ to this bit. 0 No invalid mode configuration (clock usage) interrupt occurred 1 Invalid mode configuration (clock usage) interrupt is pending
I_ICONF	**Invalid mode configuration interrupt** — This bit is set whenever a write operation to ME_<*mode*>_MC registers with invalid mode configuration is attempted. It is cleared by writing a ‘1’ to this bit. 0 No invalid mode configuration interrupt occurred 1 Invalid mode configuration interrupt is pending
I_IMODE	**Invalid mode interrupt** — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a ‘1’ to this bit. 0 No invalid mode interrupt occurred 1 Invalid mode interrupt is pending
I_SAFE	***SAFE* mode interrupt** — This bit is set whenever the chip enters ***SAFE*** mode on hardware requests generated in the system. It is cleared by writing a ‘1’ to this bit. 0 No ***SAFE*** mode interrupt occurred 1 ***SAFE*** mode interrupt is pending
I_MTC	**Mode transition complete interrupt** — This bit is set whenever the mode transition process completes (S_MTRANS transits from 1 to 0). It is cleared by writing a ‘1’ to this bit. This mode transition interrupt bit will not be set while entering low-power modes ***HALT0***, or ***STOP0***. 0 No mode transition complete interrupt occurred 1 Mode transition complete interrupt is pending

Interrupt Mask Register (ME_IM)

Figure 52. Interrupt Mask Register (ME_IM)

Address 0xC3FD_C010 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	M_ICONF_CU	M_ICONF	M_IMODE	M_SAFE	M_MTC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register controls whether an event generates an interrupt or not.

Table 51. Interrupt Mask Register (ME_IM) Field Descriptions

Field	Description
M_ICONF_CU	**Invalid mode configuration (clock usage) interrupt mask** 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled
M_ICONF	**Invalid mode configuration interrupt mask** 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled
M_IMODE	**Invalid mode interrupt mask** 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled
M_SAFE	***SAFE* mode interrupt mask** 0 ***SAFE*** mode interrupt is masked 1 ***SAFE*** mode interrupt is enabled
M_MTC	**Mode transition complete interrupt mask** 0 Mode transition complete interrupt is masked 1 Mode transition complete interrupt is enabled

Invalid Mode Transition Status Register (ME_IMTS)

Figure 53. Invalid Mode Transition Status Register (ME_IMTS)

Address 0xC3FD_C014 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status bits for the possible causes of an invalid mode interrupt.

Table 52. Invalid Mode Transition Status Register (ME_IMTS) Field Descriptions

Field	Description
S_MTI	**Mode Transition Illegal status** — This bit is set whenever a new mode is requested while some other mode transition process is active (S_MTRANS is '1'). Please refer to *Section 6.4.5: Mode Transition Interrupts* for the exceptions to this behavior. It is cleared by writing a '1' to this bit. 0 Mode transition requested is not illegal 1 Mode transition requested is illegal
S_MRI	**Mode Request Illegal status** — This bit is set whenever the target mode requested is not a valid mode with respect to current mode. It is cleared by writing a '1' to this bit. 0 Target mode requested is not illegal with respect to current mode 1 Target mode requested is illegal with respect to current mode
S_DMA	**Disabled Mode Access status** — This bit is set whenever the target mode requested is one of those disabled modes determined by ME_ME register. It is cleared by writing a '1' to this bit. 0 Target mode requested is not a disabled mode 1 Target mode requested is a disabled mode
S_NMA	**Non-existing Mode Access status** — This bit is set whenever the target mode requested is one of those non existing modes determined by ME_ME register. It is cleared by writing a '1' to this bit. 0 Target mode requested is an existing mode 1 Target mode requested is a non-existing mode
S_SEA	***SAFE* Event Active status** — This bit is set whenever the chip is in ***SAFE*** mode, ***SAFE*** event bit is pending and a new mode requested other than ***RESET***/***SAFE*** modes. It is cleared by writing a '1' to this bit. 0 No new mode requested other than ***RESET***/***SAFE*** while ***SAFE*** event is pending 1 New mode requested other than ***RESET***/***SAFE*** while ***SAFE*** event is pending

Debug Mode Transition Status Register (ME_DMTS)

Figure 54. Debug Mode Transition Status Register (ME_DMTS)

Address 0xC3FD_C018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PREVIOUS_MODE				0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	VREG_CSRC_SC	CSRC_CSRC_SC	IRC_SC	SCSRC_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	CDP_PRPH_96_127	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of different factors which influence mode transitions. It is used to give an indication of why a mode transition indicated by **ME_GS.S_MTRANS** may be taking longer than expected.

*Note: The **ME_DMTS** register does not indicate whether a mode transition is ongoing. Therefore, some **ME_DMTS** bits may still be asserted after the mode transition has completed.*

Table 53. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions

Field	Description
PREVIOUS_MODE	Previous **chip mode** — These bits show the mode in which the chip was prior to the latest change to the current mode. 0000 ***RESET*** 0001 ***TEST*** 0010 ***SAFE*** 0011 ***DRUN*** 0100 ***RUN0*** 0101 ***RUN1*** 0110 ***RUN2*** 0111 ***RUN3*** 1000 ***HALT0*** 1001 reserved 1010 ***STOP0*** 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved
MPH_BUSY	MC_ME/MC_PCU Handshake Busy indicator — This bit is set if the MC_ME has requested a mode change from the MC_PCU and the MC_PCU has not yet responded. It is cleared when the MC_PCU has responded. 0 Handshake is not busy 1 Handshake is busy
PMC_PROG	MC_PCU Mode Change in Progress indicator — This bit is set if the MC_PCU is in the process of powering up or down power domains. It is cleared when all power-up/down processes have completed. 0 Power-up/down transition is not in progress 1 Power-up/down transition is in progress
CORE_DBG	Processor is in Debug mode indicator — This bit is set while the processor is in debug mode. 0 The processor is not in debug mode 1 The processor is in debug mode
SMR	***SAFE*** mode request from MC_RGM is active indicator — This bit is set if a hardware ***SAFE*** mode request has been triggered. It is cleared when the hardware ***SAFE*** mode request has been cleared. 0 A ***SAFE*** mode request is not active 1 A ***SAFE*** mode request is active
VREG_CSRC_SC	Main VREG dependent Clock Source State Change during mode transition indicator — This bit is set when a clock source which depends on the main voltage regulator to be powered-up is requested to change its power up/down state. It is cleared when the clock source has completed its state change. 0 No state change is taking place 1 A state change is taking place
CSRC_CSRC_SC	(Other) Clock Source dependent Clock Source State Change during mode transition indicator — This bit is set when a clock source which depends on another clock source to be powered-up is requested to change its power up/down state. It is cleared when the clock source has completed its state change. 0 No state change is taking place 1 A state change is taking place

Table 53. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
IRC_SC	IRC State Change during mode transition indicator — This bit is set when the internal RC oscillator is requested to change its power up/down state. It is cleared when the internal RC oscillator has completed its state change. 0 No state change is taking place 1A state change is taking place
SCSRC_SC	Secondary Clock Sources State Change during mode transition indicator — This bit is set when a secondary clock source is requested to change its power up/down state. It is cleared when all secondary system clock sources have completed their state changes. (A 'secondary clock source' is a clock source other than IRC.) 0 No state change is taking place 1A state change is taking place
SYSCLK_SW	System Clock Switching pending status — 0 No system clock source switching is pending 1A system clock source switching is pending
DFLASH_SC	DFLASH State Change during mode transition indicator — This bit is set when the DFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place 1A state change is taking place
CFLASH_SC	CFLASH State Change during mode transition indicator — This bit is set when the CFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place 1A state change is taking place
CDP_PRPH_0_143	Clock Disable Process Pending status for Peripherals 0…143(1) — This bit is set when any peripheral has been requested to have its clock disabled. It is cleared when all the peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral
CDP_PRPH_96_127	Clock Disable Process Pending status for Peripherals 96…127(1)— This bit is set when any peripheral appearing in **ME_PS3** has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64…95(1) — This bit is set when any peripheral appearing in **ME_PS2** has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral

Table 53. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
CDP_PRPH_32_63	Clock Disable Process Pending status for Peripherals 32…63[1] — This bit is set when any peripheral appearing in **ME_PS1** has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral
CDP_PRPH_0_31	Clock Disable Process Pending status for Peripherals 0…31[1] — This bit is set when any peripheral appearing in **ME_PS0** has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1Clock disabling is pending for at least one peripheral

1. Peripheral *n* corresponds to the **ME_PCTL*n*** register. Please refer to *Table 45* for the **ME_PCTL*n*** locations actually occupied, which in turn indicates which peripherals are reported in the **ME_DMTS** register.

RESET Mode Configuration Register (ME_RESET_MC)

Figure 55. RESET Mode Configuration Register (ME_RESET_MC)

Address 0xC3FD_C020 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***RESET*** mode. Please refer to *Table 54* for details.

TEST Mode Configuration Register (ME_TEST_MC)

Figure 56. TEST Mode Configuration Register (ME_TEST_MC)

Address 0xC3FD_C024 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***TEST*** mode. Please refer to *Table 54* for details.

Note: Byte write accesses are not allowed to this register.

SAFE Mode Configuration Register (ME_SAFE_MC)

Figure 57. SAFE Mode Configuration Register (ME_SAFE_MC)

Address 0xC3FD_C028 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***SAFE*** mode. Please refer to *Table 54* for details.

Note: Byte write accesses are not allowed to this register.

DRUN Mode Configuration Register (ME_DRUN_MC)

Figure 58. DRUN Mode Configuration Register (ME_DRUN_MC)

Address 0xC3FD_C02C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***DRUN*** mode. Please refer to *Table 54* for details.

Note: Byte write accesses are not allowed to this register.

RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC)

Figure 59. RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC)

Address 0xC3FD_C030 - 0xC3FD_C03C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***RUN0…3*** modes. Please refer to *Table 54* for details.

Note: Byte write accesses are not allowed to this register.

HALT0 Mode Configuration Register (ME_HALT0_MC)

Figure 60. HALT0 Mode Configuration Register (ME_HALT0_MC)

Address 0xC3FD_C040 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***HALT0*** mode. Please refer to *Table 54* for details.

Note: Byte write accesses are not allowed to this register.

STOP0 Mode Configuration Register (ME_STOP0_MC)

Figure 61. STOP0 Mode Configuration Register (ME_STOP0_MC)

Address 0xC3FD_C048 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLL_0ON	XOSCON	IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during ***STOP0*** mode. Please refer to *Table 54* for details.

Note: Byte write accesses are not allowed to this register.

Table 54. Mode Configuration Registers (ME_<*mode*>_MC) Field Descriptions

Field	Description
PDO	**I/O output power-down control** — This bit controls the output power-down of I/Os. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In ***SAFE***/***TEST*** modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In ***STOP0*** mode, only the pad power sequence driver is disabled, but the state of the output remains functional.
MVRON	**Main voltage regulator control** — This bit specifies whether main voltage regulator is switched off or not while entering this mode. 1 Main voltage regulator is switched on
DFLAON	**Data flash power-down control** — This bit specifies the operating mode of the data flash after entering this mode. 00 reserved 01 Data flash is in power-down mode 10 reserved 11 Data flash is in normal mode
CFLAON	Code flash **power-down control** — This bit specifies the operating mode of the code flash after entering this mode. 00 reserved 01 Code flash is in power-down mode 10 Code flash is in low-power mode 11 Code flash is in normal mode
FMPLL_0ON	**system PLL control** 0 system PLL is switched off 1 system PLL is switched on
XOSCON	**external oscillator control** 0 external oscillator is switched off 1 external oscillator is switched on

Table 54. Mode Configuration Registers (ME_<*mode*>_MC) Field Descriptions (continued)

Field	Description
IRCON	**internal RC oscillator control** 0 internal RC oscillator is switched off 1 internal RC oscillator is switched on
SYSCLK	**System clock switch control** — These bits specify the system clock to be used by the system. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 PCS 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled in ***TEST*** mode, reserved in all other modes

Peripheral Status Register 0 (ME_PS0)

Figure 62. Peripheral Status Register 0 (ME_PS0)

Address 0xC3FD_C060 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						S_SafetyPort		S_FlexRay							S_FlexCAN1	S_FlexCAN0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
								S_DSPI4	S_DSPI3	S_DSPI2	S_DSPI1	S_DSPI0				
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 55* for details.

Peripheral Status Register 1 (ME_PS1)

Figure 63. Peripheral Status Register 1 (ME_PS1)

Address 0xC3FD_C064 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_CRC1	0	S_CRC0	0	0	0	0	0	0	0	0	S_LIN_FLEX1	S_LIN_FLEX0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	S_eTimer1	S_eTimer0	0	0	S_CTU0	0	0	S_ADC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 55* for details.

Peripheral Status Register 2 (ME_PS2)

Figure 64. Peripheral Status Register 2 (ME_PS2)

Address 0xC3FD_C068 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_PIT_RTI	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 55* for details.

Peripheral Status Register 3 (ME_PS3)

Figure 65. Peripheral Status Register 3 (ME_PS3)

Address 0xC3FD_C06C — Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	S_CORE1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 55* for details.

Table 55. Peripheral Status Registers (ME_PS*n*) Field Descriptions

Field	Description
S_<*periph*>	**Peripheral status** — These bits specify the current status of each peripheral which is controlled by the MC_ME. 0 Peripheral is frozen 1 Peripheral is active

Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Figure 66. Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Address 0xC3FD_C080 - 0xC3FD_C09C — Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

These registers configure eight different types of peripheral behavior during run modes.

Table 56. Run Peripheral Configuration Registers (ME_RUN_PC0…7) Field Descriptions

Field	Description
RUN3	**Peripheral control during *RUN3*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
RUN2	**Peripheral control during *RUN2*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
RUN1	**Peripheral control during *RUN1*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
RUN0	**Peripheral control during *RUN0*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
DRUN	**Peripheral control during *DRUN*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
SAFE	**Peripheral control during *SAFE*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
TEST	**Peripheral control during *TEST*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
RESET	**Peripheral control during *RESET*** 0 Peripheral is frozen with clock gated 1 Peripheral is active

Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

Figure 67. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

Address 0xC3FD_C0A0 - 0xC3FD_C0BC Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	STOP0	0	HALT0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

These registers configure eight different types of peripheral behavior during non-run modes.

Table 57. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) Field Descriptions

Field	Description
STOP0	**Peripheral control during *STOP0*** 0 Peripheral is frozen with clock gated 1 Peripheral is active
HALT0	**Peripheral control during *HALT0*** 0 Peripheral is frozen with clock gated 1 Peripheral is active

Peripheral Control Registers (ME_PCTL*n*)

Figure 68. Peripheral Control Registers (ME_PCTLn)

Address 0xC3FD_C0C0 - 0xC3FD_C14F Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	0	DBG_F	LP_CFG			RUN_CFG		
W								
Reset	0	0	0	0	0	0	0	0

These registers select the configurations during run and non-run modes for each peripheral. Please refer to *Table 45* for information on which **ME_PCTL*n*** locations are actually occupied. The unoccupied locations contain a read-only byte value of 0x00.

Table 58. Peripheral Control Registers (ME_PCTLn) Field Descriptions

Field	Description
DBG_F	Peripheral control in debug mode — This bit controls the state of the peripheral in debug mode 0 Peripheral state depends on RUN_CFG/LP_CFG bits and the chip mode 1 Peripheral is frozen if not already frozen in chip modes. *Note: This feature is useful to freeze the peripheral state while entering debug. For example, this may be used to prevent a reference timer from running while making a debug accesses.*

Table 58. Peripheral Control Registers (ME_PCTLn) Field Descriptions (continued)

Field	Description
LP_CFG	**Peripheral configuration select for non-run modes** — These bits associate a configuration as defined in the **ME_LP_PC0...7** registers to the peripheral. 000 Selects ME_LP_PC0 configuration 001 Selects ME_LP_PC1 configuration 010 Selects ME_LP_PC2 configuration 011 Selects ME_LP_PC3 configuration 100 Selects ME_LP_PC4 configuration 101 Selects ME_LP_PC5 configuration 110 Selects ME_LP_PC6 configuration 111 Selects ME_LP_PC7 configuration
RUN_CFG	**Peripheral configuration select for run modes** — These bits associate a configuration as defined in the **ME_RUN_PC0...7** registers to the peripheral. 000 Selects ME_RUN_PC0 configuration 001 Selects ME_RUN_PC1 configuration 010 Selects ME_RUN_PC2 configuration 011 Selects ME_RUN_PC3 configuration 100 Selects ME_RUN_PC4 configuration 101 Selects ME_RUN_PC5 configuration 110 Selects ME_RUN_PC6 configuration 111 Selects ME_RUN_PC7 configuration

*After modifying any of the **ME_RUN_PC0...7**, **ME_LP_PC0...7**, and **ME_PCTLn** registers, software must request a mode change and wait for the mode change to be completed before entering debug mode in order to have consistent behavior between the peripheral clock control process and the clock status reporting in the **ME_PSn** registers.*

6.4 Functional Description

6.4.1 Mode Transition Request

The transition from one mode to another mode is normally handled by software by accessing the mode control register **ME_MCTL**. But the in case of special events, the mode transition can be automatically managed by hardware. In order to switch from one mode to another, the application should access the **ME_MCTL** register twice by writing

- the first time with the value of the key (0x5AF0) into the **KEY** bit field and the required target mode into the **TARGET_MODE** bit field,
- and the second time with the inverted value of the key (0xA50F) into the **KEY** bit field and the required target mode into the **TARGET_MODE** bit field.

Once a valid mode transition request is detected, the target mode configuration information is loaded from the corresponding **ME_<*mode*>_MC** register.The mode transition request may require a number of cycles depending on the programmed configuration, and software should check the **S_CURRENT_MODE** bit field and the **S_MTRANS** bit of the global status register **ME_GS** to verify when the mode has been correctly entered and the transition process has completed. For a description of valid mode requests, please refer to *Section 6.4.5 Mode Transition Interrupts*.

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately but on the next request to enter this mode. This means

that transition requests such as ***RUN0…3*** → ***RUN0…3***, ***DRUN*** → ***DRUN***, ***SAFE*** → ***SAFE***, and ***TEST*** → ***TEST*** are considered valid mode transition requests. As soon as the mode request is accepted as valid, the **S_MTRANS** bit is set till the status in the **ME_GS** register matches the configuration programmed in the respective **ME_<*mode*>_MC** register.

*Note: It is recommended that software poll the **S_MTRANS** bit in the **ME_GS** register after requesting a transition to **HALT0** or **STOP0** modes.*

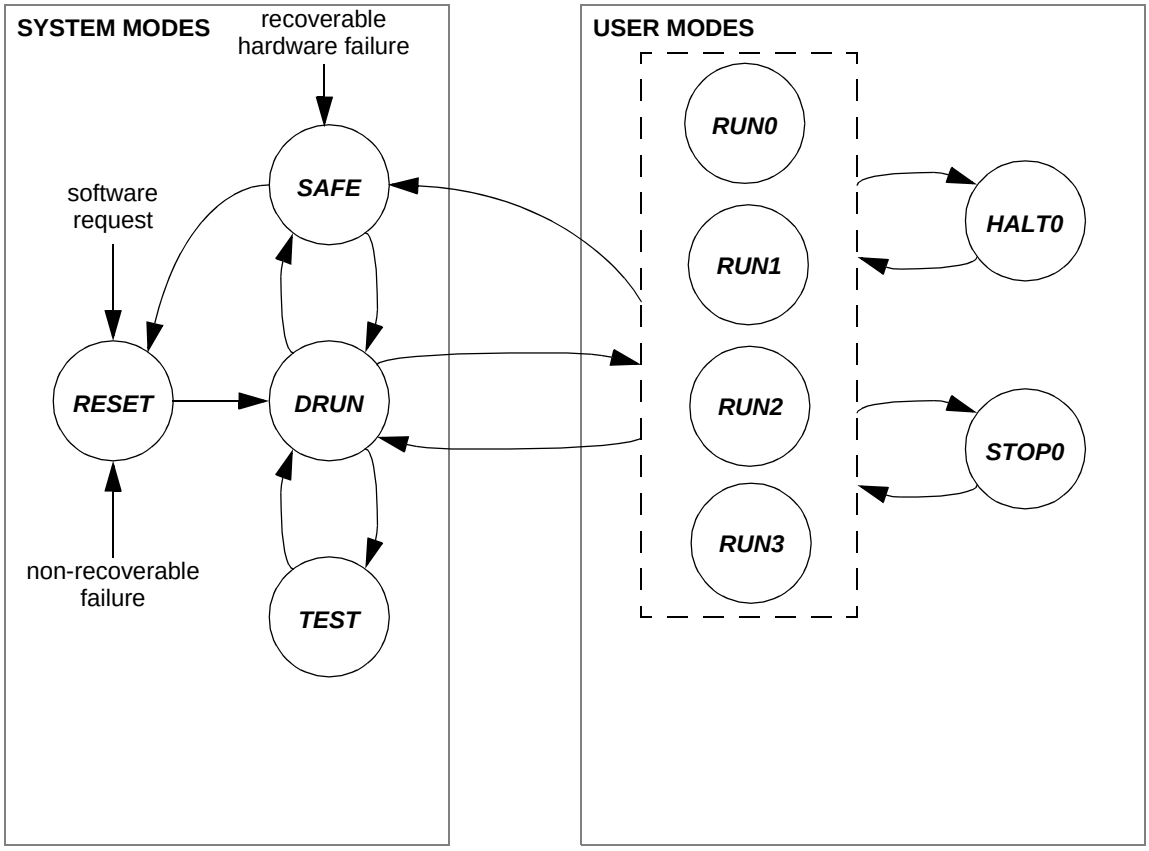


Figure 69. MC_ME Mode Diagram

6.4.2 Modes Details

RESET Mode

The chip enters this mode on the following events:

- from ***SAFE***, ***DRUN***, ***RUN0…3***, or ***TEST*** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “0000” for a ‘functional’ reset
- from any mode due to a system reset by the MC_RGM because of some non-recoverable hardware failure in the system (see the MC_RGM chapter for details)

Transition to this mode is instantaneous, and the system remains in this mode until the reset sequence is finished. The mode configuration information for this mode is provided by the **ME_RESET_MC** register. This mode has a pre-defined configuration, and the IRC is selected as the system clock.

DRUN Mode

The chip enters this mode on the following events:

- automatically from ***RESET*** mode after completion of the reset sequence
- from ***RUN0…3***, ***SAFE***, or ***TEST*** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “0011”

As soon as any of the above events has occurred, a ***DRUN*** mode transition request is generated. The mode configuration information for this mode is provided by the **ME_DRUN_MC** register. In this mode, the flashes, all clock sources, and the system clock configuration can be controlled by software as required. After system reset, the software execution starts with the default configuration selecting the IRC as the system clock.

This mode is intended to be used by software

- to initialize all registers as per the system needs

Note: *Software must ensure that the code executes from RAM before changing to this mode if the flashes are configured to be in the low-power or power-down state in this mode.*

SAFE Mode

The chip enters this mode on the following events:

- from ***DRUN***, ***RUN0…3***, or ***TEST*** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “0010”
- from any mode except ***RESET*** due to a ***SAFE*** mode request generated by the MC_RGM because of some potentially recoverable hardware failure in the system (see the MC_RGM chapter for details)

*Note: If a hardware **SAFE** mode request occurs during **RESET**, depending on the timing of the **SAFE** mode request, **SAFE** mode may be entered immediately after the normal completion of the reset sequence or several system clock cycles after **DRUN** entry. The **SAFE** mode request does not have any influence on the execution of the reset sequence itself.*

As soon as any of the above events has occurred, a ***SAFE*** mode transition request is generated. The mode configuration information for this mode is provided by the **ME_SAFE_MC** register. This mode has a pre-defined configuration, and the IRC is selected as the system clock.

If the ***SAFE*** mode is requested by software while some other mode transition process is ongoing, the new target mode becomes the ***SAFE*** mode regardless of other pending requests or new requests during the mode transition. No new mode request made during a transition to the SAFE mode will cause an invalid mode interrupt.

*Note: If software requests to change to the **SAFE** mode and then requests to change back to the parent mode before the mode transition is completed, the chip’s final mode after mode transition will be the **SAFE** mode.*

As long as a ***SAFE*** event is active, the system remains in the ***SAFE*** mode, and any software mode request during this time is ignored and lost.

This mode is intended to be used by software

- to assess the severity of the cause of failure and then to either
  - re-initialize the chip via the ***DRUN*** mode, or
  - completely reset the chip via the ***RESET*** mode.

If the outputs of the system I/Os need to be forced to a high impedance state upon entering this mode, the **PDO** bit of the **ME_SAFE_MC** register should be set. The input levels remain unchanged.

TEST Mode

The chip enters this mode on the following events:

- from the ***DRUN*** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “0001”

As soon as any of the above events has occurred, a ***TEST*** mode transition request is generated. The mode configuration information for this mode is provided by the **ME_TEST_MC** register. Except for the main voltage regulator, all resources of the system are configurable in this mode. The system clock to the whole system can be stopped by programming the **SYSCLK** bit field to “1111”, and in this case, the only way to exit this mode is via a chip reset.

This mode is intended to be used by software

- to execute software test routines

Note: *Software must ensure that the code executes from RAM before changing to this mode if the flashes are configured to be in the low-power or power-down state in this mode.*

RUN0…3 Modes

The chip enters one of these modes on the following events:

- from the ***DRUN***, ***SAFE***, or another ***RUN0…3*** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “0100…0111”
- from the ***HALT0*** mode due to an interrupt event
- from the ***STOP0*** mode due to an interrupt or wakeup event

As soon as any of the above events has occurred, a ***RUN0…3*** mode transition request is generated. The mode configuration information for these modes is provided by the **ME_RUN0…3_MC** registers. In these modes, the flashes, all clock sources, and the system clock configuration can be controlled by software as required.

These modes are intended to be used by software

- to execute application routines

Note: *Software must ensure that the code executes from RAM before changing to this mode if the flashes are configured to be in the low-power or power-down state in this mode.*

HALT0 Mode

The chip enters this mode on the following events:

- from one of the ***RUN0…3*** modes when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “1000”.

As soon as any of the above events has occurred, a ***HALT0*** mode transition request is generated. The mode configuration information for this mode is provided by **ME_HALT0_MC** register. This mode is quite configurable, and the **ME_HALT0_MC** register should be programmed according to the system needs. The flashes can be put in low-power or power-down mode as needed. If there is a ***HALT0*** mode request while an interrupt request is active, the transition to ***HALT0*** is aborted with the resultant mode being the

current mode, ***SAFE*** (on ***SAFE*** mode request), or ***DRUN*** (on reset), and an invalid mode interrupt is not generated.

This mode is intended as a first-level low-power mode with

- the core clock frozen
- only a few peripherals running

and to be used by software

- to wait until it is required to do something and then to react quickly (i.e., within a few system clock cycles of an interrupt event)

*Note: It is good practice for software to ensure that the **S_MTRANS** bit in the **ME_GS** register has been cleared on **HALT0** mode exit to ensure that the previous **RUN0…3** mode configuratoin has been fully restored before executing critical code.*

STOP0 Mode

The chip enters this mode on the following events:

- from one of the ***RUN0…3*** modes when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “1010”.

As soon as any of the above events has occurred, a ***STOP0*** mode transition request is generated. The mode configuration information for this mode is provided by the **ME_STOP0_MC** register. This mode is fully configurable, and the **ME_STOP0_MC** register should be programmed according to the system needs.

The flashes can be put in power-down mode as needed. If there is a ***STOP0*** mode request while any interrupt or wakeup event is active, the transition to ***STOP0*** is aborted with the resultant mode being the current mode, ***SAFE*** (on ***SAFE*** mode request), or ***DRUN*** (on reset), and an invalid mode interrupt is not generated.

This can be used as an advanced low-power mode with the core clock frozen and almost all peripherals stopped.

This mode is intended as an advanced low-power mode with

- the core clock frozen
- almost all peripherals stopped

and to be used by software

- to wait until it is required to do something with no need to react quickly (e.g., allow for system clock source to be re-started)

This mode can be used to stop all clock sources, except internal RC oscillator clock, and thus preserve the chip status. When exiting the ***STOP0*** mode, the internal RC oscillator clock is selected as the system clock until the target clock is available.

Note: *It is good practice for software to ensure that the **S_MTRANS** bit in the **ME_GS** register has been cleared on **STOP0** mode exit to ensure that the previous **RUN0…3** mode configuratoin has been fully restored before executing critical code.*

6.4.3 Mode Transition Process

The process of mode transition follows the following steps in a pre-defined manner depending on the current chip mode and the requested target mode. In many cases of

mode transition, not all steps need to be executed based on the mode control information, and some steps may not be applicable according to the mode definition itself.

Target Mode Request

The target mode is requested by accessing the **ME_MCTL** register with the required keys. This mode transition request by software must be a valid request satisfying a set of pre-defined rules to initiate the process. If the request fails to satisfy these rules, it is ignored, and the **TARGET_MODE** bit field is not updated. An optional interrupt can be generated for invalid mode requests. Refer to *Section 6.4.5: Mode Transition Interrupts* for details.

In the case of mode transitions occurring because of hardware events such as a reset, a ***SAFE*** mode request, or interrupt requests and wakeup events to exit from low-power modes, the **TARGET_MODE** bit field of the **ME_MCTL** register is automatically updated with the appropriate target mode. The mode change process start is indicated by the setting of the mode transition status bit **S_MTRANS** of the **ME_GS** register.

A ***RESET*** mode requested via the **ME_MCTL** register is passed to the MC_RGM, which generates a global system reset and initiates the reset sequence. The ***RESET*** mode request has the highest priority, and the MC_ME is kept in the ***RESET*** mode during the entire reset sequence.

The ***SAFE*** mode request has the next highest priority after reset. It can be generated either by software via the **ME_MCTL** register from all software running modes including ***DRUN***, ***RUN0...3***, and ***TEST*** or by the MC_RGM after the detection of system hardware failures, which may occur in any mode.

Target Mode Configuration Loading

On completion of the *Target Mode Request* step, the target mode configuration from the **ME_*<target mode>*_MC** register is loaded to start the resources (voltage sources, clock sources, flashes, pads, etc.) control process.

An overview of resource control possibilities for each mode is shown in . A '√' indicates that a given resource is configurable for a given mode.

Table 59. MC_ME Resource Control Overview

Resource	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0...3***	***HALT0***	***STOP0***
IRC	on	√ on	on	on	on	on	on
XOSC	off	√ off	off	√ off	√ off	√ off	√ off
FMPLL_0	off	√ off	off	√ off	√ off	√ off	off
CFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down

Table 59. MC_ME Resource Control Overview (continued)

Resource	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0…3***	***HALT0***	***STOP0***
DFLASH	normal	√ normal	normal	√ normal	√ normal	√ normal	√ power-down
MVREG	on	on	on	on	on	√ on	√ on
PDO	off	√ off	√ on	off	off	off	√ off

Peripheral Clocks Disable

On completion of the *Target Mode Request* step, the MC_ME requests each peripheral to enter its stop mode when:

- the peripheral is configured to be disabled via the target mode, the peripheral configuration registers **ME_RUN_PC0…7** and **ME_LP_PC0…7**, and the peripheral control registers **ME_PCTL*n***

Note: The MC_ME automatically requests peripherals to enter their stop modes if the power domains in which they are residing are to be turned off due to a mode change. However, it is good practice for software to ensure that those peripherals that are to be powered down are configured in the MC_ME to be frozen.

Each peripheral acknowledges its stop mode request after closing its internal activity. The MC_ME then disables the corresponding clock(s) to this peripheral.

In the case of a ***SAFE*** mode transition request, the MC_ME does not wait for the peripherals to acknowledge the stop requests. The ***SAFE*** mode clock gating configuration is applied immediately regardless of the status of the peripherals' stop acknowledges.

Please refer to *Section 6.4.6 Peripheral Clock Gating* for more details.

Each peripheral that may block or disrupt a communication bus to which it is connected ensures that these outputs are forced to a safe or recessive state when the chip enters the ***SAFE*** mode.

Processor Low-Power Mode Entry

If, on completion of the *Peripheral Clocks Disable* step, the mode transition is to the ***HALT0*** mode, the MC_ME requests the processor to enter its halted state. The processor acknowledges its halt state request after completing all outstanding bus transactions.

If, on completion of the *Peripheral Clocks Disable* step, the mode transition is to the ***STOP0*** mode, the MC_ME requests the processor to enter its stopped state. The processor acknowledges its stop state request after completing all outstanding bus transactions.

Processor and System Memory Clock Disable

If, on completion of the *Processor Low-Power Mode Entry* step, the mode transition is to the ***HALT0*** or ***STOP0*** mode and the processor is in its appropriate halted or stopped state, the MC_ME disables the processor and system memory clocks to achieve further power saving.

The clocks to the processor and system memory are unaffected while transitioning between software running modes such as ***DRUN***, ***RUN0…3***, and ***SAFE***.

Warning: *Clocks to the whole chip including the processor and system memory can be disabled in **TEST** mode.*

Clock Sources Switch-On

On completion of the *Processor Low-Power Mode Entry* step, the MC_ME switches on all clock sources based on the **<*clock source*>ON** bits of the **ME_<*current mode*>_MC** and **ME_<*target mode*>_MC** registers. The following clock sources are switched on at this step:

- the internal RC oscillator
- the external oscillator
- the system PLL

The clock sources that are required by the target mode are switched on. The duration required for the output clocks to be stable depends on the type of source, and all further steps of mode transition depending on one or more of these clocks waits for the stable status of the respective clocks. The availability status of these clocks is updated in the **S_<*clock source*>** bits of **ME_GS** register.

The clock sources which need to be switched off are unaffected during this process in order to not disturb the system clock which might require one of these clocks before switching to a different target clock.

Flash Modules Switch-On

On completion of the step, if one or more of the flashes needs to be switched to normal mode from its low-power or power-down mode based on the **CFLAON** and **DFLAON** bit fields of the **ME_<*current mode*>_MC** and **ME_<*target mode*>_MC** registers, the MC_ME requests the flash to exit from its low-power/power-down mode. When the flashes are available for access, the **S_CFLA** and **S_DFLA** bit fields of the **ME_GS** register are updated to “11” by hardware.

Warning: *It is illegal to switch the CFLASH from low-power mode to power-down mode and from power-down mode to low-power mode. The MC_ME, however, does not prevent this nor does it flag it.*

Pad Outputs-On

On completion of the step, if the **PDO** bit of the **ME_<*target mode*>_MC** register is cleared, then

- all pad outputs are enabled to return to their previous state
- the I/O pads power sequence driver is switched on

Peripheral Clocks Enable

Based on the current and target chip modes, the peripheral configuration registers **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and the peripheral control registers **ME_PCTL*n***, the MC_ME enables the clocks for selected modules as required. This step is executed only after the process is completed.

Processor and Memory Clock Enable

If the mode transition is from any of the low-power modes ***HALT0*** or ***STOP0*** to ***RUN0…3***, the clocks to the processor and system memory are enabled. The process of enabling these clocks is executed only after the *Flash Modules Switch-On* process is completed.

Processor Low-Power Mode Exit

If the mode transition is from any of the low-power modes ***HALT0*** or***STOP0*** to ***RUN0…3***, the MC_ME requests the processor to exit from its halted or stopped state. This step is executed only after the *Processor and Memory Clock Enable* process is completed.

System Clock Switching

Based on the **SYSCLK** bit field of the **ME_<*current mode*>_MC** and **ME_<*target mode*>_MC** registers, if the target and current system clock configurations differ, the following method is implemented for clock switching.

- The target clock configuration for the IRC takes effect only after the **S_IRC** bit of the **ME_GS** register is set by hardware (i.e., the internal RC oscillator has stabilized).
- The target clock configuration for the XOSC takes effect only after the **S_XOSC** bit of the **ME_GS** register is set by hardware (i.e., the external oscillator has stabilized).
- The target clock configuration for the FMPLL_0 PCS takes effect only after the **S_FMPLL_0** bit of the **ME_GS** register is set by hardware (i.e., the system PLL has stabilized).
- If the clock is to be disabled, the **SYSCLK** bit field should be programmed with “1111”. This is possible only in the***TEST*** mode.

The current system clock configuration can be observed by reading the **S_SYSCLK** bit field of the **ME_GS** register, which is updated after every system clock switching. Until the target clock is available, the system uses the previous clock configuration.

System clock switching starts only after

- the *Clock Sources Switch-On* process has completed if the target system clock source is one of the following:
  - the internal RC oscillator
  - the system PLL
- the *Peripheral Clocks Disable* process has completed in order not to change the system clock frequency before peripherals close their internal activities

An overview of system clock source selection possibilities for each mode is shown in *Table 60*. A ‘√’ indicates that a given clock source is selectable for a given mode.

Table 60. MC_ME System Clock Selection Overview

System Clock Source	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0…3***	***HALT0***	***STOP0***
IRC	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)		
XOSC		√		√	√	√	√

Table 60. MC_ME System Clock Selection Overview (continued)

System Clock Source	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0...3***	***HALT0***	***STOP0***
FMPLL_0 PCS		√		√	√	√	
system clock is disabled		√(1)					

1. disabling the system clock during ***TEST*** mode will require a reset in order to exit ***TEST*** mode

Pad Switch-Off

If the **PDO** bit of the **ME_<*target mode*>_MC** register is '1' then

- the outputs of the pads are forced to the high impedance state if the target mode is ***SAFE*** or ***TEST***

This step is executed only after the *Peripheral Clocks Disable* process has completed.

Clock Sources (with no Dependencies) Switch-Off

Based on the chip mode and the **<*clock source*>ON** bits of the **ME_<*mode*>_MC** registers, if a given clock source is to be switched off and no other clock source needs it to be on, the MC_ME requests the clock source to power down and updates its availability status bit **S_<*clock source*>** of the **ME_GS** register to '0'. The following clock sources switched off at this step:

- the system PLL

This step is executed only after the *System Clock Switching* process has completed.

Clock Sources (with Dependencies) Switch-Off

Based on the chip mode and the **<*clock source*>ON** bits of the **ME_<*mode*>_MC** registers, if a given clock source is to be switched off and all clock sources which need this clock source to be on have been switched off, the MC_ME requests the clock source to power down and updates its availability status bit **S_<*clock source*>** of the **ME_GS** register to '0'. The following clock sources switched off at this step:

- the external oscillator

This step is executed only after

- the *System Clock Switching* process has completed in order not to lose the current system clock during mode transition
- the *Clock Sources (with no Dependencies) Switch-Off* process has completed in order to, for example, prevent unwanted lock transitions

Flash Switch-Off

Based on the **CFLAON** and **DFLAON** bit fields of the **ME_<*current mode*>_MC** and **ME_<*target mode*>_MC** registers, if any of the flashes is to be put in its low-power or power-down mode, the MC_ME requests the flash to enter the corresponding power mode and waits for the flash to acknowledge. The exact power mode status of the flashes is

updated in the **S_CFLA** and **S_DFLA** bit fields of the **ME_GS** register. This step is executed only when the *Processor and System Memory Clock Disable* process has completed.

Current Mode Update

The current mode status bit field **S_CURRENT_MODE** of the **ME_GS** register is updated with the target mode bit field **TARGET_MODE** of the **ME_MCTL** register when :

- all the updated status bits in the **ME_GS** register match the configuration specified in the **ME_*<target mode>*_MC** register
- power sequences are done
- clock disable/enable process is finished
- processor low-power mode (halt/stop) entry and exit processes are finished

Note: ***SAFE*** *mode entry does not wait for the clock disable/enable process to finish. It only waits for the* ***ME_GS.S_RC*** *bit to be set. This is to ensure that the* ***SAFE*** *mode is entered as quickly as possible.*

Software can monitor the mode transition status by reading the **S_MTRANS** bit of the **ME_GS** register. The mode transition latency can differ from one mode to another depending on the resources' availability before the new mode request and the target mode's requirements.

If a mode transition is taking longer to complete than is expected, the **ME_DMTS** register can indicate which process is still in progress.

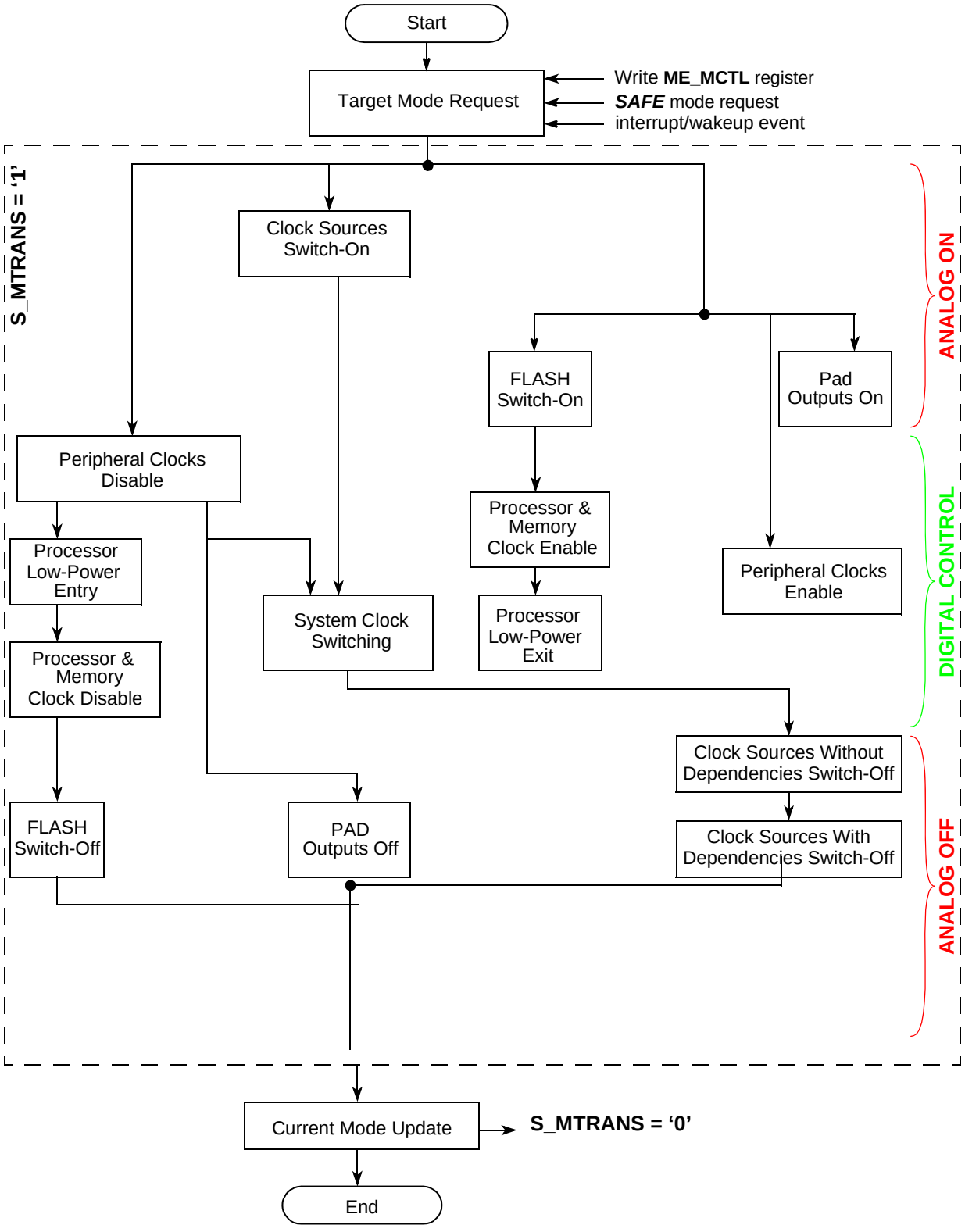


Figure 70. MC_ME Transition Diagram

6.4.4 Protection of Mode Configuration Registers

While programming the mode configuration registers **ME_<*mode*>_MC**, the following rules must be respected. Otherwise, the write operation is ignored and an invalid mode configuration interrupt may be generated.

- If the IRC is selected as the system clock, IRC must be on.
- If the XOSC clock is selected as the system clock, OSC must be on.
- If the FMPLL_0 PCS clock is selected as the system clock, PLL must be on.
- If FMPLL_0 is on, XOSC must also be on.

Note: *Software must ensure that clock sources with dependencies other than those mentioned above are switched on as needed. There is no automatic protection mechanism to check this in the MC_ME.*

- Configuration “00” for the **CFLAON** bit field is reserved.
- Configuration “00” for the **DFLAON** bit field is reserved.
- Configuration “10” for the **DFLAON** bit field is reserved.
- Configuration “11” for the **DFLAON** bit field is reserved if the **CFLAON** bit field is not “11”.
- System clock configurations marked as ‘reserved’ may not be selected.
- Configuration “1111” for the **SYSCLK** bit field is allowed only for the***TEST*** mode, and only in this case may all system clock sources be turned off.

Warning: *If the system clock is stopped during **TEST** mode, the chip can exit only via a system reset.*

6.4.5 Mode Transition Interrupts

The MC_ME provides interrupts for incorrectly configuring a mode, requesting an invalid mode transition, indicating a ***SAFE*** mode transition not due to a software request, and indicating when a mode transition has completed.

Invalid Mode Configuration Interrupt

Whenever a write operation is attempted to the **ME_<*mode*>_MC** registers violating the protection rules mentioned in the *Section 6.4.4: Protection of Mode Configuration Registers*, the interrupt pending bit **I_ICONF** of the **ME_IS** register is set and an interrupt request is generated if the mask bit **M_ICONF** of the **ME_IM** register is ‘1’.

In addition, during a mode transition, if a clock source has been configured in the **ME_<*target mode*>_MC** register to be off and a peripheral requiring this clock source to be on has been enabled via the **ME_RUN_PC0…7**/**ME_LP_PC0…7** and **ME_PCTL*n*** registers, the interrupt pending bit **I_ICONF_CU** of the **ME_IS** register is set and an iterrupt request is generated if the mask bit **M_ICONF_CU** of the **ME_IM** register is ‘1’.

Invalid Mode Transition Interrupt

The mode transition request is considered invalid under the following conditions:

- If the system is in the ***SAFE*** mode and the ***SAFE*** mode request from MC_RGM is active, and if the target mode requested is other than ***RESET*** or ***SAFE***, then this new

mode request is considered to be invalid, and the **S_SEA** bit of the **ME_IMTS** register is set.

- If the **TARGET_MODE** bit field of the **ME_MCTL** register is written with a value different from the specified mode values (i.e., a non-existing mode), an invalid mode transition event is generated. When such a non existing mode is requested, the **S_NMA** bit of the **ME_IMTS** register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the **ME_MCTL** register.
- If some of the chip modes are disabled as programmed in the **ME_ME** register, their respective configurations are considered reserved, and any access to the **ME_MCTL** register with those values results in an invalid mode transition request. When such a disabled mode is requested, the **S_DMA** bit of the **ME_IMTS** register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the **ME_MCTL** register.
- If the target mode is not a valid mode with respect to the current mode, the mode request illegal status bit **S_MRI** of the **ME_IMTS** register is set. This condition is detected only when the proper key mechanism is followed while writing the **ME_MCTL** register. Otherwise, the write operation is ignored.
- If further new mode requests occur while a mode transition is in progress (the **S_MTRANS** bit of the **ME_GS** register is '1'), the mode transition illegal status bit **S_MTI** of the **ME_IMTS** register is set. This condition is detected only when the proper key mechanism is followed while writing the **ME_MCTL** register. Otherwise, the write operation is ignored.

*Note: As the causes of invalid mode transitions may overlap at the same time, the priority implemented for invalid mode transition status bits of the **ME_IMTS** register in the order from highest to lowest is **S_SEA**, **S_NMA**, **S_DMA**, **S_MRI**, and **S_MTI**.*

As an exception, the mode transition request is not considered as invalid under the following conditions:

- A new request is allowed to enter the ***RESET*** or ***SAFE*** mode irrespective of the mode transition status.
- As the exit of ***HALT0*** and ***STOP0*** modes depends on the interrupts of the system which can occur at any instant, these requests to return to ***RUN0…3*** modes are always valid.
- In order to avoid any unwanted lockup of the chip modes, software can abort a mode transition by requesting the parent mode if, for example, the mode transition has not completed after a software determined 'reasonable' amount of time for whatever reason. The parent mode is the chip mode before a valid mode request was made.
- Self-transition requests (e.g., ***RUN0*** → ***RUN0***) are not considered as invalid even when the mode transition process is active (i.e., **S_MTRANS** is '1'). During the low-power mode exit process, if the system is not able to enter the respective ***RUN0…3*** mode properly (i.e., all status bits of the **ME_GS** register match with configuration bits in the **ME_<*mode*>_MC** register), then software can only request the ***SAFE*** or ***RESET*** mode. It is not possible to request any other mode or to go back to the low-power mode again.

Whenever an invalid mode request is detected, the interrupt pending bit **I_IMODE** of the **ME_IS** register is set, and an interrupt request is generated if the mask bit **M_IMODE** of the **ME_IM** register is '1'.

SAFE Mode Transition Interrupt

Whenever the system enters the ***SAFE*** mode as a result of a ***SAFE*** mode request from the MC_RGM due to a hardware failure, the interrupt pending bit **I_SAFE** of the **ME_IS** register is set, and an interrupt is generated if the mask bit **M_SAFE** of **ME_IM** register is '1' .

The ***SAFE*** mode interrupt pending bit can be cleared only when the ***SAFE*** mode request is deasserted by the MC_RGM (see the MC_RGM chapter for details on how to clear a ***SAFE*** mode request). If the system is already in ***SAFE*** mode, any new ***SAFE*** mode request by the MC_RGM also sets the interrupt pending bit **I_SAFE**. However, the ***SAFE*** mode interrupt pending bit is not set when the ***SAFE*** mode is entered by a software request (i.e., programming of **ME_MCTL** register).

Mode Transition Complete interrupt

Whenever the system fully completes a mode transition (i.e., the **S_MTRANS** bit of **ME_GS** register transits from '1' to '0'), the interrupt pending bit **I_MTC** of the **ME_IS** register is set, and an interrupt request is generated if the mask bit **M_MTC** of the **ME_IM** register is '1'. The interrupt bit **I_MTC** is not set when entering low-power modes ***HALT0*** and ***STOP0*** in order to avoid the same event requesting the immediate exit of these low-power modes.

6.4.6 Peripheral Clock Gating

During all chip modes, each peripheral can be associated with a particular clock gating policy determined by two groups of peripheral configuration registers.

The run peripheral configuration registers **ME_RUN_PC0…7** are chosen only during the software running modes ***DRUN***, ***TEST***, ***SAFE***, and ***RUN0…3***. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Run configuration selection for each peripheral is done by the **RUN_CFG** bit field of the **ME_PCTL*n*** registers.

The low-power peripheral configuration registers **ME_LP_PC0…7** are chosen only during the low-power modes ***HALT0*** and ***STOP0***. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Low-power configuration selection for each peripheral is done by the **LP_CFG** bit field of the **ME_PCTL*n*** registers.

Any modifications to the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL*n*** registers do not affect the clock gating behavior until a new mode transition request is generated.

Whenever the processor enters a debug session during any mode, the following occurs for each peripheral:

- The clock is gated if the **DBG_F** bit of the associated **ME_PCTL*n*** register is set. Otherwise, the peripheral clock gating status depends on the **RUN_CFG** and **LP_CFG** bits. Any further modifications of the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL*n*** registers during a debug session will take affect immediately without requiring any new mode request.

6.4.7 Application Example

Figure 71 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

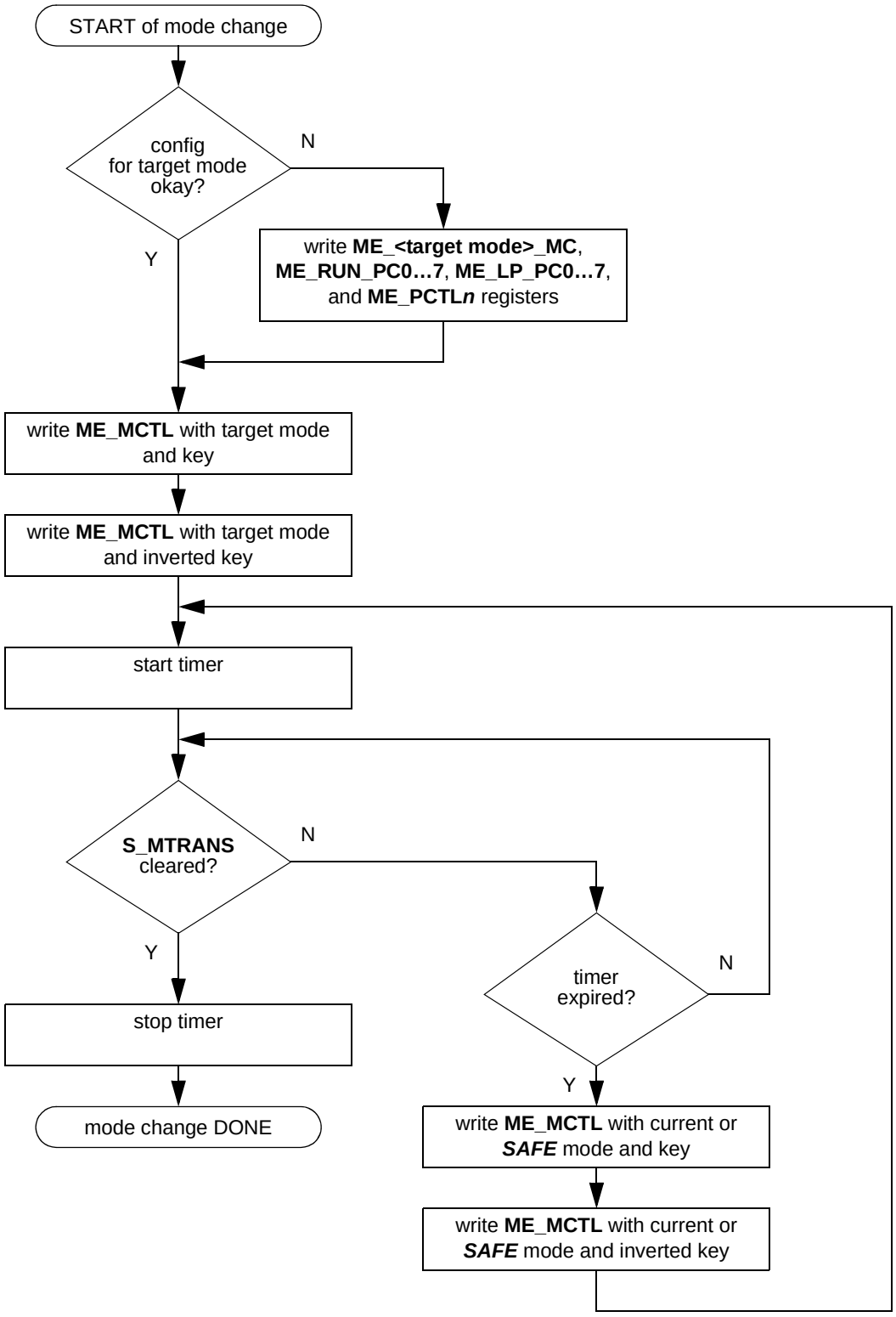


Figure 71. MC_ME Application Example Flow Diagram

7 Power Control Unit (MC_PCU)

7.1 Introduction

7.1.1 Overview

The power control unit (MC_PCU) acts as a bridge for mapping the PMU peripheral to the MC_PCU address space.

Figure 72 depicts the MC_PCU block diagram.

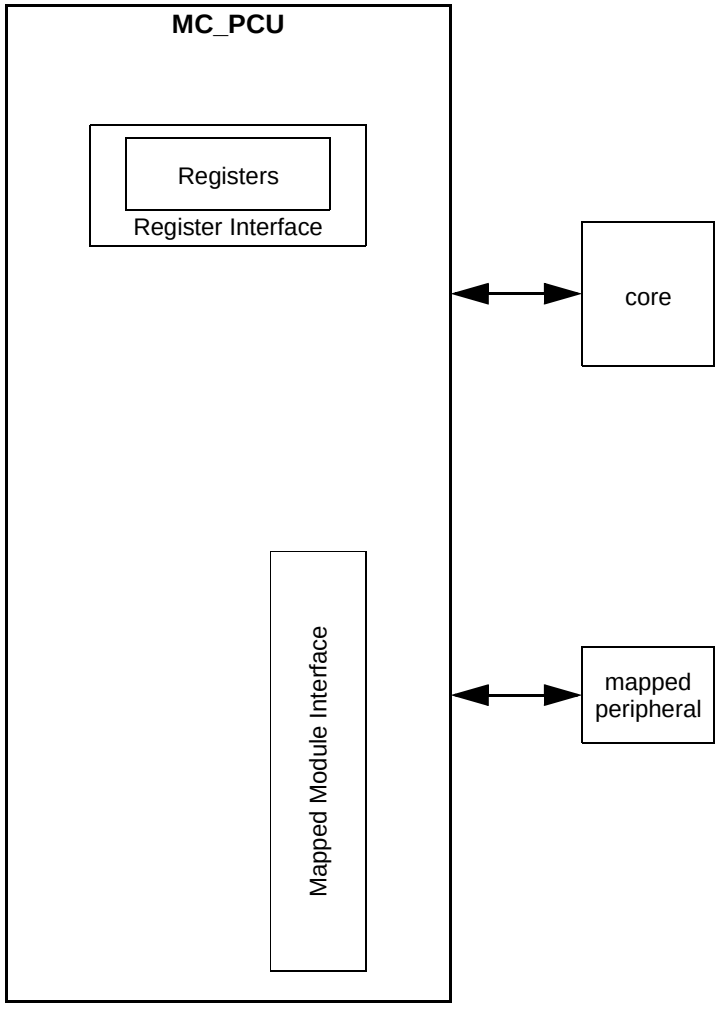


Figure 72. MC_PCU Block Diagram

7.1.2 Features

The MC_PCU includes the following features:

- maps the PMU registers to the MC_PCU address space

7.2 External Signal Description

The MC_PCU has no connections to any external pins.

7.3 Memory Map and Register Definition

7.3.1 Memory Map

Table 61. MC_PCU memory map

Offset from PCU_BASE (0xC3FE_8000)	Register	Location
0x0000	PCU_PCONF0—Power Domain #0 Configuration register	*on page 205*
0x0004–0x003F	Reserved	
0x0040	PCU_PSTAT—Power Domain Status register	*on page 206*
0x0044–0x007F	Reserved	
0x0080	VREG_CTL—Voltage Regulator Control register	*on page 1055*
0x0084	VREG_STATUS—Voltage Regulator Status register	*on page 1056*
0x0044–0x7FFF	Reserved	

Note: Accesses to unused registers as well as write accesses to read-only registers do not change register content, and can cause a transfer error.

Table 62. MC_PCU registers

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_8000	PCU_PCONF0	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	STBY0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_8004 … 0xC3FE_803F	reserved																	
0xC3FE_8040	PCU_PSTAT	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PD0
		W																
0x044 … 0x07F	reserved																	

Table 62. MC_PCU registers (continued) (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_8080	VREG_CTL[1]	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V_LVD_MASK
		W																
0xC3FE_8084	VREG_STATUS[2]	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V_LVD_STATUS
		W																
0xC3FE_8088 ... 0xC3FE_BFFF	reserved																	

1. See *Section : Voltage Regulator Control Register (VREG_CTL).*
2. See *Section : Voltage Regulator Status register (VREG_STATUS).*

7.3.2 Register Descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the PD0 field of the PCU_PSTAT register may be accessed as a word at address 0xC3FE_8040, as a half-word at address 0xC3FE_8042, or as a byte at address 0xC3FE_8043.

Power Domain #0 Configuration Register (PCU_PCONF0)

Figure 73. Power Domain #0 Configuration Register (PCU_PCONF0)

Address 0xC3FE_8000 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1

This register defines for power domain #0 whether it is on or off in each chip mode. As power domain #0 is the always-on power domain (and includes the MC_PCU), none of its bits are programmable. This register is available for completeness reasons.

Table 63. Power Domain Configuration Register Field Descriptions

Field	Description
RST	Power domain control during ***RESET*** mode 0 Power domain off 1 Power domain on
TEST	Power domain control during ***TEST*** mode 0 Power domain off 1 Power domain on
SAFE	Power domain control during ***SAFE*** mode 0 Power domain off 1 Power domain on
DRUN	Power domain control during ***DRUN*** mode 0 Power domain off 1 Power domain on
RUN0	Power domain control during ***RUN0*** mode 0 Power domain off 1 Power domain on
RUN1	Power domain control during ***RUN1*** mode 0 Power domain off 1 Power domain on
RUN2	Power domain control during ***RUN2*** mode 0 Power domain off 1 Power domain on

Table 63. Power Domain Configuration Register Field Descriptions (continued)

Field	Description
RUN3	Power domain control during ***RUN3*** mode 0 Power domain off 1 Power domain on
HALT0	Power domain control during ***HALT0*** mode 0 Power domain off 1 Power domain on
STOP0	Power domain control during ***STOP0*** mode 0 Power domain off 1 Power domain on
STBY0	Power domain control during ***STANDBY0*** mode 0 Power domain off 1 Power domain on

Power Domain Status Register (PCU_PSTAT)

Figure 74. Power Domain Status Register (PCU_PSTAT)

Address 0xC3FE_8040 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

This register reflects the power status of all available power domains.

Table 64. Power Domain Status Register (PCU_PSTAT) Field Descriptions

Field	Description
PD*n*	Power status for power domain #*n* 0Power domain is inoperable 1Power domain is operable

8 Reset Generation Module (MC_RGM)

8.1 Introduction

8.1.1 Overview

The reset generation module (MC_RGM) centralizes the different reset sources and manages the reset sequence of the chip. It provides a register interface and the reset sequencer. Various registers are available to monitor and control the chip reset sequence. The reset sequencer is a state machine which controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and controls the reset signals generated in the system.

Figure 75 shows the MC_RGM block diagram.

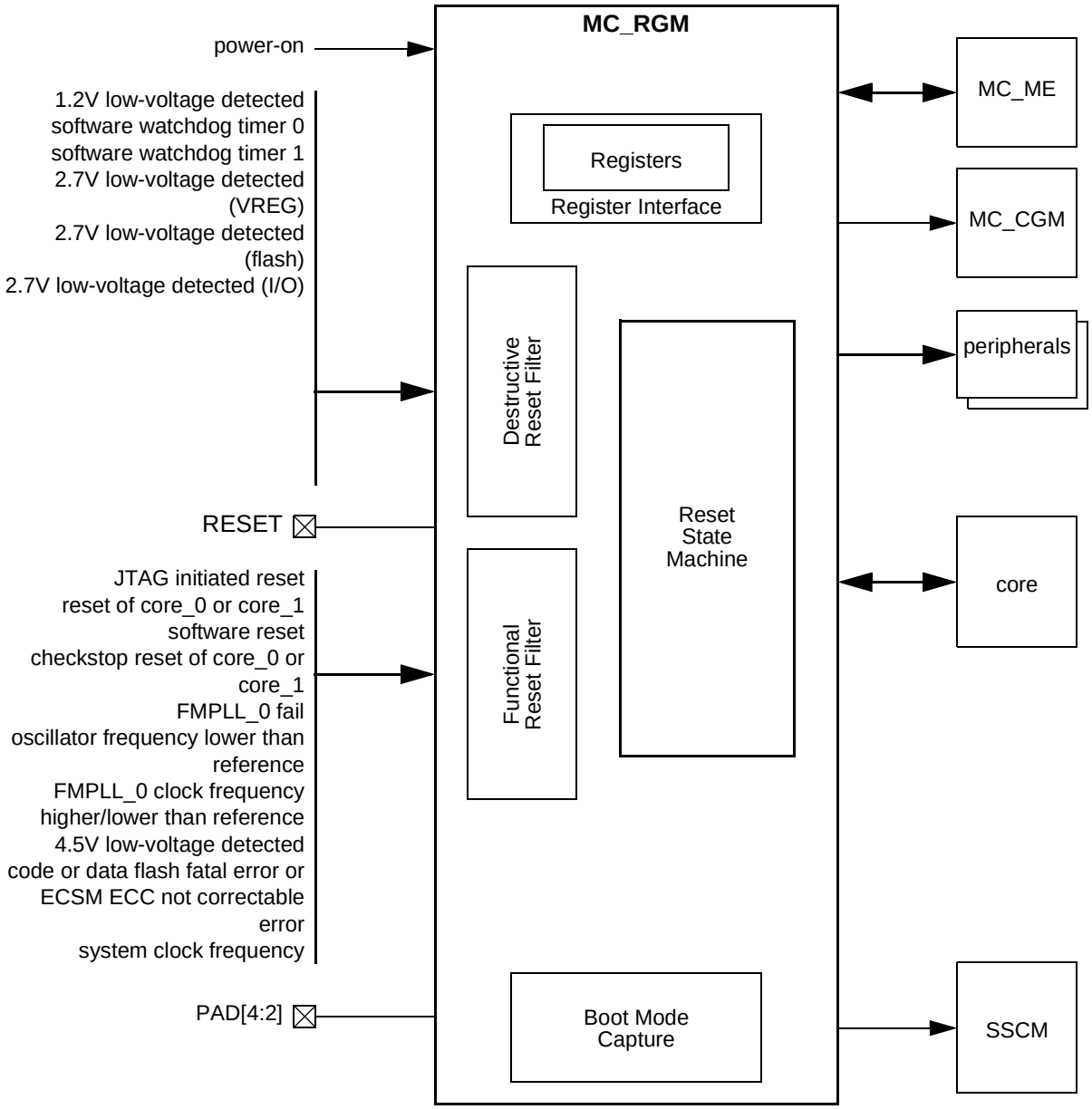


Figure 75. MC_RGM Block Diagram

8.1.2 Features

The MC_RGM contains the functionality for the following features:

- ‘destructive’ resets management
- ‘functional’ resets management
- signalling of reset events after each reset sequence (reset status flags)
- conversion of reset events to SAFE mode or interrupt request events
- short reset sequence configuration
- bidirectional reset behavior configuration
- boot mode capture on $\overline{\text{RESET}}$ deassertion

8.1.3 Reset Sources

The different reset sources are organized into two families: ‘destructive’ and ‘functional’.

- A ‘destructive’ reset source is associated with an event related to a critical - usually hardware - error or dysfunction. When a ‘destructive’ reset event occurs, the full reset sequence is applied to the chip starting from PHASE0. This resets the full chip ensuring a safe start-up state for both digital and analog modules, and the memory content must be considered to be unknown. ‘Destructive’ resets are
  - power-on reset
  - 1.2V low-voltage detected
  - software watchdog timer 0
  - software watchdog timer 1
  - 2.7V low-voltage detected (VREG)
  - 2.7V low-voltage detected (flash)
  - 2.7V low-voltage detected (I/O)
- A ‘functional’ reset source is associated with an event related to a less-critical - usually non-hardware - error or dysfunction. When a ‘functional’ reset event occurs, a partial reset sequence is applied to the chip starting from PHASE1. In this case, most digital modules are reset normally, while the state of analog modules or specific digital modules (e.g., debug modules, flash modules) is preserved. ‘Functional’ resets are
  - external reset
  - JTAG initiated reset
  - reset of core_0 or core_1
  - software reset
  - checkstop reset of core_0 or core_1
  - FMPLL_0 fail
  - oscillator frequency lower than reference
  - FMPLL_0 clock frequency higher/lower than reference
  - 4.5V low-voltage detected
  - code or data flash fatal error or ECSM ECC not correctable error
  - system clock frequency higher/lower than reference

When a reset is triggered, the MC_RGM state machine is activated and proceeds through the different phases (i.e., PHASEn states). Each phase is associated with a particular chip reset being provided to the system. A phase is completed when all corresponding phase

completion gates from either the system or internal to the MC_RGM are acknowledged. The chip reset associated with the phase is then released, and the state machine proceeds to the next phase up to entering the IDLE phase. During this entire process, the MC_ME state machine is held in RESET mode. Only at the end of the reset sequence, when the IDLE phase is reached, does the MC_ME enter the DRUN mode.

Alternatively, it is possible for software to configure some reset source events to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt issued to the core (see *Section : Functional Event Reset Disable Register (RGM_FERD)* and *Section : Functional Event Alternate Request Register (RGM_FEAR)* for 'functional' resets).

8.2 External Signal Description

The MC_RGM interfaces to the bidirectional reset pin $\overline{\text{RESET}}$ and the boot mode pins PAD[4:2].

8.3 Memory Map and Register Definition

Table 65. MC_RGM Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_4000	RGM_FES	Functional Event Status	half-word	read	read/write(1)	read/write(1)	*on page 213*
0xC3FE_4002	RGM_DES	Destructive Event Status	half-word	read	read/write(1)	read/write1	*on page 214*
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	half-word	read	read/write(2)	read/write(2)	*on page 216*
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	half-word	read	read	read	*on page 217*
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	half-word	read	read/write	read/write	*on page 218*
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	half-word	read	read/write	read/write	*on page 219*
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	half-word	read	read/write	read/write	*on page 221*

1. individual bits cleared on writing '1'
2. write once: '0' = enable, '1' = disable.

Note: Any access to unused registers as well as write accesses to read-only registers will:

- *not change register content*
- *cause a transfer error*

Table 66. MC_RGM Memory Map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4000	RGM_FES / RGM_DES	R	F_EXR	0	0	0	F_CMU1_FHL	0	0	F_FLASH	F_LVD45	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
		W	w1c				w1c			w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	F_SWT1	F_SWT0		F_LVD12
		W	w1c									w1c	w1c	w1c	w1c	w1c		w1c
0xC3FE_4004	RGM_FERD / RGM_DERD	R	D_EXR	0	0		D_CMU1_FHL	0	0	0	D_LVD45	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	D_SWT1	D_SWT0	0	D_LVD12
		W																
0xC3FE_4008 … 0xC3FE_400F	reserved																	
0xC3FE_4010	RGM_FEAR	R	0	0	0	0	AR_CMU1_FHL	0	0	0	AR_LVD45	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	0	0	AR_CORE	AR_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 66. MC_RGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4014 … 0xC3FE_4017	reserved																	
0xC3FE_4018	RGM_FESS	R	SS_EXR	0	0	0	SS_CMU1_FHL	0	0	SS_FLASH	SS_LVD45	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_401C	RGM_FBRE	R	BE_EXR	0	0	0	BE_CMU1_FHL	0	0	BE_FLASH	BE_LVD45	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_401E … 0xC3FE_7FFF	reserved																	

8.3.1 Register Descriptions

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the RGM_DES[8:15] register bits may be accessed as a word at address 0xC3FE_4000, as a half-word at address 0xC3FE_4002, or as a byte at address 0xC3FE_4003.

Some fields may be read-only, and their reset value of ‘1’ or ‘0’ and the corresponding behavior cannot be changed.

Functional Event Status Register (RGM_FES)

Figure 76. Functional Event Status Register (RGM_FES)

Address 0xC3FE_4000 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	0	0	0	F_CMU1_FHL	0	0	F_FLASH	F_LVD45	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
W	w1c				w1c			w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. It can be accessed in read only in user mode. Register bits are cleared on write '1' if the triggering event has already been cleared at the source.

Note: If a 'functional' reset source is configured to generate a SAFE mode request or an interrupt request, software needs to clear the event in the source module at least three system clock cycles before it clears the associated RGM_FES status bit in order to avoid multiple SAFE mode requests or interrupts for the same event. In order to avoid having to count cycles, it is good practice for software to check whether the RGM_FES has been properly cleared, and if not, clear it again.

Table 67. Functional Event Status Register (RGM_FES) Field Descriptions

Field	Description
F_EXR	**Flag for External Reset** 0 No external reset event has occurred since either the last clear or the last destructive reset assertion 1 An external reset event has occurred
F_CMU1_FHL	**Flag for system clock frequency higher/lower than reference** 0 No system clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A system clock frequency higher/lower than reference event has occurred
F_FLASH	**Flag for code or data flash fatal error or ECSM ECC not correctable error** 0 No code or data flash fatal error or ECSM ECC not correctable error event has occurred since either the last clear or the last destructive reset assertion 1 A code or data flash fatal error or ECSM ECC not correctable error event has occurred
F_LVD45	**Flag for 4.5V low-voltage detected** 0 No 4.5V low-voltage detected event has occurred since either the last clear or the last destructive reset assertion 1 A 4.5V low-voltage detected event has occurred
F_CMU0_FHL	**Flag for FMPLL_0 clock frequency higher/lower than reference** 0 No FMPLL_0 clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A FMPLL_0 clock frequency higher/lower than reference event has occurred

Table 67. Functional Event Status Register (RGM_FES) Field Descriptions (continued)

Field	Description
F_CMU0_OLR	**Flag for oscillator frequency lower than reference** 0 No oscillator frequency lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A oscillator frequency lower than reference event has occurred
F_PLL0	**Flag for FMPLL_0 fail** 0 No FMPLL_0 fail event has occurred since either the last clear or the last destructive reset assertion 1 A FMPLL_0 fail event has occurred
F_CHKSTOP	**Flag for checkstop reset of core_0 or core_1** 0 No checkstop reset of core_0 or core_1 event has occurred since either the last clear or the last destructive reset assertion 1 A checkstop reset of core_0 or core_1 event has occurred
F_SOFT	**Flag for software reset** 0 No software reset event has occurred since either the last clear or the last destructive reset assertion 1 A software reset event has occurred
F_CORE	**Flag for reset of core_0 or core_1** 0 No reset of core_0 or core_1 event has occurred since either the last clear or the last destructive reset assertion 1 A reset of core_0 or core_1 event has occurred
F_JTAG	**Flag for JTAG initiated reset** 0 No JTAG initiated reset event has occurred since either the last clear or the last destructive reset assertion 1 A JTAG initiated reset event has occurred

Destructive Event Status Register (RGM_DES)

Figure 77. Destructive Event Status Register (RGM_DES)

Address 0xC3FE_4002 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	F_SWT1	F_SWT0	0	F_LVD12
W	w1c									w1c	w1c	w1c	w1c	w1c		w1c
POR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the status of the last asserted destructive reset sources. It can be accessed in read/write on either supervisor mode or test mode. It can be accessed in read only in user mode. Register bits are cleared on write ‘1’.

Table 68. Destructive Event Status Register (RGM_DES) Field Descriptions

Field	Description
F_POR	**Flag for Power-On reset** 0 No power-on event has occurred since the last clear 1 A power-on event has occurred
F_LVD27_IO	**Flag for 2.7V low-voltage detected (I/O)** 0 No 2.7V low-voltage detected (I/O) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V low-voltage detected (I/O) event has occurred
F_LVD27_FLASH	**Flag for 2.7V low-voltage detected (flash)** 0 No 2.7V low-voltage detected (flash) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V low-voltage detected (flash) event has occurred
F_LVD27_VREG	**Flag for 2.7V low-voltage detected (VREG)** 0 No 2.7V low-voltage detected (VREG) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V low-voltage detected (VREG) event has occurred
F_SWT1	**Flag for software watchdog timer 1** 0 No software watchdog timer 1 event has occurred since either the last clear or the last power-on reset assertion 1 A software watchdog timer 1 event has occurred
F_SWT0	**Flag for software watchdog timer 0** 0 No software watchdog timer 0 event has occurred since either the last clear or the last power-on reset assertion 1 A software watchdog timer 0 event has occurred
F_LVD12	**Flag for 1.2V low-voltage detected** 0 No 1.2V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2V low-voltage detected event has occurred

Note: The F_POR flag is automatically cleared on a 1.2 V low-voltage detected or a 2.7 V low-voltage detected. This means that if the power-up sequence is not monotonic (i.e., the voltage rises and then drops enough to trigger a low-voltage detection), the F_POR flag may not be set but instead the <register>F_LVD12 or <register>F_LVD27_VREG flag is set on exiting the reset sequence. Therefore, if the F_POR, <register>F_LVD12 or <register>F_LVD27_VREG flags are set on reset exit, software should interpret the reset cause as power-on.

Functional Event Reset Disable Register (RGM_FERD)

Figure 78. Functional Event Reset Disable Register (RGM_FERD)

Address 0xC3FE_4004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D_EXR	0	0	0	D_CMU1_FHL	0	0	D_FLASH	D_LVD45	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides dedicated bits to disable functional reset sources.When a functional reset source is disabled, the associated functional event will trigger either a SAFE mode request or an interrupt request (see *Section : Functional Event Alternate Request Register (RGM_FEAR)*). It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode. Each byte can be written only once after power-on reset.

Warning: *It is important to clear the RGM_FES register before setting any of the bits in the RGM_FERD register to '1'. Otherwise a redundant SAFE mode request or interrupt request may occur.*

Table 69. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions

Field	Description
D_EXR	**Disable External Reset** 0 An external reset event triggers a reset sequence
D_CMU1_FHL	**Disable system clock frequency higher/lower than reference** 0 A system clock frequency higher/lower than reference event triggers a reset sequence 1 A system clock frequency higher/lower than reference event generates **either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CMU1_FHL**
D_FLASH	**Disable code or data flash fatal error or ECSM ECC not correctable error** 0 A code or data flash fatal error or ECSM ECC not correctable error event triggers a reset sequence
D_LVD45	**Disable 4.5V low-voltage detected** 0 A 4.5V low-voltage detected event triggers a reset sequence 1 A 4.5V low-voltage detected event generates either **a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_LVD45**
D_CMU0_FHL	**Disable FMPLL_0 clock frequency higher/lower than reference** 0 A FMPLL_0 clock frequency higher/lower than reference event triggers a reset sequence 1 A FMPLL_0 clock frequency higher/lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CMU0_FHL

Table 69. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions (continued)

Field	Description
D_CMU0_OLR	**Disable oscillator frequency lower than reference** 0 A oscillator frequency lower than reference event triggers a reset sequence 1 A oscillator frequency lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CMU0_OLR
D_PLL0	**Disable FMPLL_0 fail** 0 A FMPLL_0 fail event triggers a reset sequence 1 A FMPLL_0 fail event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_PLL0
D_CHKSTOP	**Disable checkstop reset of core_0 or core_1** 0 A checkstop reset of core_0 or core_1 event triggers a reset sequence
D_SOFT	**Disable software reset** 0 A software reset event triggers a reset sequence
D_CORE	**Disable reset of core_0 or core_1** 0 A reset of core_0 or core_1 event triggers a reset sequence 1 A reset of core_0 or core_1 event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CORE
D_JTAG	**Disable JTAG initiated reset** 0 A JTAG initiated reset event triggers a reset sequence 1 A JTAG initiated reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_JTAG

Destructive Event Reset Disable Register (RGM_DERD)

Figure 79. Destructive Event Reset Disable Register (RGM_DERD)

Address 0xC3FE_4006 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	D_SWT1	D_SWT0	0	D_LVD12
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides dedicated bits to disable particular destructive reset sources. It can be accessed in read-only in supervisor mode, test mode, and user mode.

Table 70. Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions

Field	Description
D_LVD27_IO	**Disable 2.7V low-voltage detected (I/O)** 0 A 2.7V low-voltage detected (I/O) event triggers a reset sequence
D_LVD27_FLASH	**Disable 2.7V low-voltage detected (flash)** 0 A 2.7V low-voltage detected (flash) event triggers a reset sequence
D_LVD27_VREG	**Disable 2.7V low-voltage detected (VREG)** 0 A 2.7V low-voltage detected (VREG) event triggers a reset sequence
D_SWT1	**Disable software watchdog timer 1** 0 A software watchdog timer 1 event triggers a reset sequence
D_SWT0	**Disable software watchdog timer 0** 0 A software watchdog timer 0 event triggers a reset sequence
D_LVD12	**Disable 1.2V low-voltage detected** 0 A 1.2V low-voltage detected event triggers a reset sequence

Functional Event Alternate Request Register (RGM_FEAR)

Figure 80. Functional Event Alternate Request Register (RGM_FEAR)

Address 0xC3FE_4010 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	AR_CMU1_FHL	0	0	0	AR_LVD45	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	0	0	AR_CORE	AR_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines an alternate request to be generated when a reset on a functional event has been disabled. The alternate request can be either a SAFE mode request to MC_ME or an interrupt request to the system. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode.

Table 71. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions

Field	Description
AR_CMU1_FHL	**Alternate Request for system clock frequency higher/lower than reference** 0 Generate a ***SAFE*** mode request on a system clock frequency higher/lower than reference event if the reset is disabled 1Generate an interrupt request on a system clock frequency higher/lower than reference event if the reset is disabled
AR_LVD45	**Alternate Request for 4.5V low-voltage detected** 0 Generate a ***SAFE*** mode request on a 4.5V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 4.5V low-voltage detected event if the reset is disabled

Table 71. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions

Field	Description
AR_CMU0_FHL	**Alternate Request for FMPLL_0 clock frequency higher/lower than reference** 0 Generate a SAFE mode request on a FMPLL_0 clock frequency higher/lower than reference event if the reset is disabled 1Generate an interrupt request on a FMPLL_0 clock frequency higher/lower than reference event if the reset is disabled
AR_CMU0_OLR	**Alternate Request for oscillator frequency lower than reference** 0 Generate a SAFE mode request on a oscillator frequency lower than reference event if the reset is disabled 1Generate an interrupt request on a oscillator frequency lower than reference event if the reset is disabled
AR_PLL0	**Alternate Request for FMPLL_0 fail** 0 Generate a SAFE mode request on a FMPLL_0 fail event if the reset is disabled 1 Generate an interrupt request on a FMPLL_0 fail event if the reset is disabled
AR_CORE	**Alternate Request for reset of core_0 or core_1** 0 Generate a SAFE mode request on a reset of core_0 or core_1 event if the reset is disabled 1 Generate an interrupt request on a reset of core_0 or core_1 event if the reset is disabled
AR_JTAG	**Alternate Request for JTAG initiated reset** 0 Generate a SAFE mode request on a JTAG initiated reset event if the reset is disabled 1 Generate an interrupt request on a JTAG initiated reset event if the reset is disabled

Functional Event Short Sequence Register (RGM_FESS)

Figure 81. Functional Event Short Sequence Register (RGM_FESS)

Address 0xC3FE_4018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS_EXR	0	0	0	SS_CMU1_FHL	0	0	SS_FLASH	SS_LVD45	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines which reset sequence will be done when a functional reset sequence is triggered. The functional reset sequence can either start from PHASE1 or from PHASE3, skipping PHASE1 and PHASE2.

Note: This could be useful for fast reset sequence, for example to skip flash reset.

It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 72. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions

Field	Description
SS_EXR	**Short Sequence for External Reset** 0 The reset sequence triggered by an **external** reset event will start from ***PHASE1***
SS_CMU1_FHL	**Short Sequence for system clock frequency higher/lower than reference** 0 The reset sequence triggered by a system clock frequency higher/lower than reference event will start from PHASE1 1 The reset sequence triggered by a system clock frequency higher/lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2
SS_FLASH	**Short Sequence for code or data flash fatal error or ECSM ECC not correctable error** 0 The reset sequence triggered by a code or data flash fatal error or ECSM ECC not correctable error event will start from PHASE1
SS_LVD45	**Short Sequence for 4.5V low-voltage detected** 0 The reset sequence triggered by a 4.5V low-voltage detected event will start from PHASE1 1 The reset sequence triggered by a 4.5V low-voltage detected event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CMU0_FHL	**Short Sequence for FMPLL_0 clock frequency higher/lower than reference** 0 The reset sequence triggered by a FMPLL_0 clock frequency higher/lower than reference event will start from PHASE1 1 The reset sequence triggered by a FMPLL_0 clock frequency higher/lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CMU0_OLR	**Short Sequence for oscillator frequency lower than reference** 0 The reset sequence triggered by a oscillator frequency lower than reference event will start from PHASE1 1 The reset sequence triggered by a oscillator frequency lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2
SS_PLL0	**Short Sequence for FMPLL_0 fail** 0 The reset sequence triggered by a FMPLL_0 fail event will start from PHASE1 1 The reset sequence triggered by a FMPLL_0 fail event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CHKSTOP	**Short Sequence for checkstop reset of core_0 or core_1** 0 The reset sequence triggered by a checkstop reset of core_0 or core_1 event will start from **PHASE1**
SS_SOFT	**Short Sequence for software reset** 0 The reset sequence triggered by a software reset event will start from **PHASE1**
SS_CORE	**Short Sequence for reset of core_0 or core_1** 0 The reset sequence triggered by a reset of core_0 or core_1 event will start from PHASE1 1 The reset sequence triggered by a reset of core_0 or core_1 event will start from PHASE3, skipping PHASE1 and PHASE2
SS_JTAG	**Short Sequence for JTAG initiated reset** 0 The reset sequence triggered by a JTAG initiated reset event will start from PHASE1 1 The reset sequence triggered by a JTAG initiated reset event will start from PHASE3, skipping PHASE1 and PHASE2

Note: This register is reset on any enabled 'destructive' or 'functional' reset event.

Functional Bidirectional Reset Enable Register (RGM_FBRE)

Figure 82. Functional Bidirectional Reset Enable Register (RGM_FBRE)

Address 0xC3FE_401C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BE_EXR	0	0	0	BE_CMU1_FHL	0	0	BE_FLASH	BE_LVD45	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register enables the generation of an external reset on functional reset. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.reset

Table 73. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_EXR	**Bidirectional Reset Enable for External Reset** 0$\overline{\text{RESET}}$ is asserted on an **external** reset event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on an **external** reset event
BE_CMU1_FHL	**Bidirectional Reset Enable for system clock frequency higher/lower than reference** 0$\overline{\text{RESET}}$ is asserted on a system clock frequency higher/lower than reference event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a system clock frequency higher/lower than reference event
BE_FLASH	**Bidirectional Reset Enable for code or data flash fatal error or ECSM ECC not correctable error** 0$\overline{\text{RESET}}$ is asserted on a code or data flash fatal error or ECSM ECC not correctable error event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a code or data flash fatal error or ECSM ECC not correctable error event
BE_LVD45	**Bidirectional Reset Enable for 4.5V low-voltage detected** 0$\overline{\text{RESET}}$ is asserted on a 4.5V low-voltage detected event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a 4.5V low-voltage detected event
BE_CMU0_FHL	**Bidirectional Reset Enable for FMPLL_0 clock frequency higher/lower than reference** 0$\overline{\text{RESET}}$ is asserted on a FMPLL_0 clock frequency higher/lower than reference event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a FMPLL_0 clock frequency higher/lower than reference event
BE_CMU0_OLR	**Bidirectional Reset Enable for oscillator frequency lower than reference** 0$\overline{\text{RESET}}$ is asserted on a oscillator frequency lower than reference event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a oscillator frequency lower than reference event
BE_PLL0	**Bidirectional Reset Enable for FMPLL_0 fail** 0$\overline{\text{RESET}}$ is asserted on a FMPLL_0 fail event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a FMPLL_0 fail event

Table 73. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_CHKSTOP	**Bidirectional Reset Enable for checkstop reset of core_0 or core_1** 0$\overline{\text{RESET}}$ is asserted on a checkstop reset of core_0 or core_1 event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a checkstop reset of core_0 or core_1 event
BE_SOFT	**Bidirectional Reset Enable for software reset** 0$\overline{\text{RESET}}$ is asserted on a software reset event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a software reset event
BE_CORE	**Bidirectional Reset Enable for reset of core_0 or core_1** 0$\overline{\text{RESET}}$ is asserted on a reset of core_0 or core_1 event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a reset of core_0 or core_1 event
BE_JTAG	**Bidirectional Reset Enable for JTAG initiated reset** 0$\overline{\text{RESET}}$ is asserted on a JTAG initiated reset event if the reset is enabled 1$\overline{\text{RESET}}$ is not asserted on a JTAG initiated reset event

8.4 Functional Description

8.4.1 Reset State Machine

The main role of MC_RGM is the generation of the reset sequence which ensures that the correct parts of the chip are reset based on the reset source event. This is summarized in *Table 74*.

Table 74. MC_RGM Reset Implications

Source	What Gets Reset	External Reset Assertion(1)	Boot Mode Capture
power-on reset	all	yes	yes
‘destructive’ resets	all except some clock/reset management	yes	yes
external reset	all except some clock/reset management and debug	programmable (2)	yes
‘functional’ resets	all except some clock/reset management and debug	programmable[2]	programmable (3)
shortened ‘functional’ resets(4)	flip-flops except some clock/reset management	programmable[2]	programmable[3]

1. ‘external reset assertion’ means that the $\overline{\text{RESET}}$ pin is asserted by the MC_RGM until the end of reset PHASE3
2. the assertion of the external reset is controlled via the RGM_FBRE register
3. the boot mode is captured if the external reset is asserted
4. the short sequence is enabled via the RGM_FESS register

Note: JTAG logic has its own independent reset control and is not controlled by the MC_RGM in any way.

The reset sequence is comprised of five phases managed by a state machine, which ensures that all phases are correctly processed through waiting for a minimum duration and

until all processes that need to occur during that phase have been completed before proceeding to the next phase.

The state machine used to produce the reset sequence is shown in *Figure 83*.

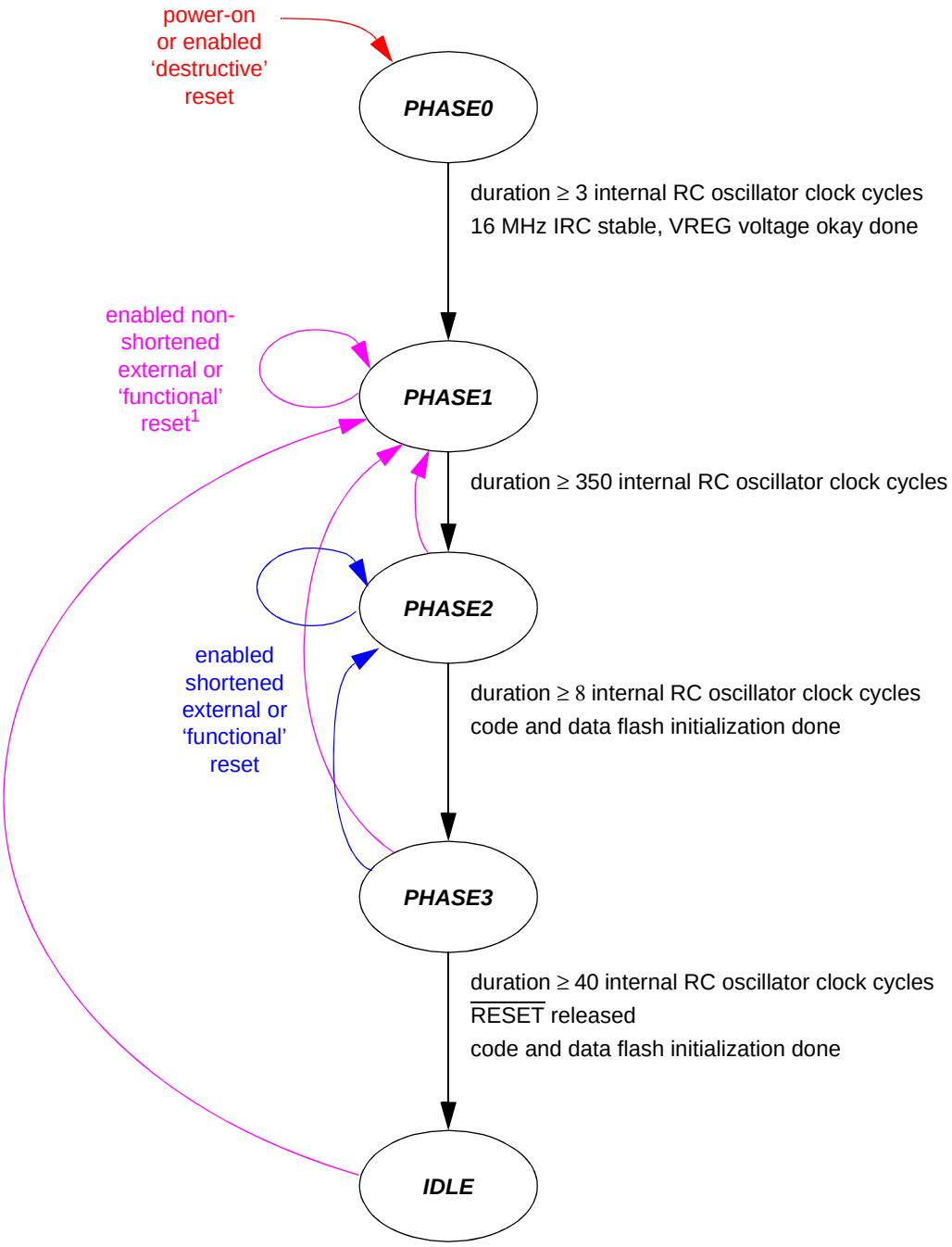


Figure 83. MC_RGM State Machine

PHASE0 Phase

This phase is entered immediately from any phase on a power-on or enabled ‘destructive’ reset event. The reset state machine exits PHASE0 and enters PHASE1 on verification of the following:

- all enabled ‘destructive’ resets have been processed
- all processes that need to be done in PHASE0 are completed
  - 16 MHz IRC stable, VREG voltage okay
- a minimum of 3 internal RC oscillator clock cycles have elapsed since power-up completion and the last enabled ‘destructive’ reset event

PHASE1 Phase

This phase is entered either on exit from PHASE0 or immediately from PHASE2, PHASE3, or IDLE on a non-masked external or ‘functional’ reset event if it has not been configured to trigger a ‘short’ sequence. The reset state machine exits PHASE1 and enters PHASE2 on verification of the following:

- all enabled, non-shortened ‘functional’ resets have been processed
- a minimum of 350 internal RC oscillator clock cycles have elapsed since the last enabled external or non-shortened ‘functional’ reset event

PHASE2 Phase

This phase is entered on exit from PHASE1. The reset state machine exits PHASE2 and enters PHASE3 on verification of the following:

- all processes that need to be done in PHASE2 are completed
  - code and data flash initialization
- a minimum of 8 internal RC oscillator clock cycles have elapsed since entering ***PHASE2***

PHASE3 Phase

This phase is a entered either on exit from PHASE2 or immediately from IDLE on an enabled, shortened ‘functional’ reset event. The reset state machine exits PHASE3 and enters IDLE on verification of the following:

- all processes that need to be done in PHASE3 are completed
  - code and data flash initialization
- a minimum of 40 internal RC oscillator clock cycles have elapsed since the last enabled, shortened ‘functional’ reset event

IDLE Phase

This is the final phase and is entered on exit from PHASE3. When this phase is reached, the MC_RGM releases control of the chip to the core and waits for new reset events that can trigger a reset sequence.

8.4.2 Destructive Resets

A ‘destructive’ reset indicates that an event has occurred after which critical register or memory content can no longer be guaranteed.

The status flag associated with a given ‘destructive’ reset event (RGM_DES.F_<destructive reset> bit) is set when the ‘destructive’ reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software’s responsibility to determine which reset source is the most critical for the application.

The chip’s low-voltage detector threshold ensures that, when 1.2V low-voltage detected is enabled, the supply is sufficient to have the destructive event correctly propagated through the digital logic. Therefore, if a given ‘destructive’ reset is enabled, the MC_RGM ensures that the associated reset event will be correctly triggered to the full system. However, if the given ‘destructive’ reset is disabled and the voltage goes below the digital functional threshold, functionality can no longer be ensured, and the reset may or may not be asserted.

An enabled ‘destructive’ reset will trigger a reset sequence starting from the beginning of PHASE0.

8.4.3 External Reset

The MC_RGM manages the external reset coming from $\overline{\text{RESET}}$. The detection of a falling edge on $\overline{\text{RESET}}$ will start the reset sequence from the beginning of PHASE1. By default $\overline{\text{RESET}}$ pin is an internal pull-down configuration.

The status flag associated with the external reset falling edge event (RGM_FES.F_EXR bit) is set when the external reset is asserted and the power-on reset is not asserted.

The external reset can optionally be disabled by writing bit RGM_FERD.D_EXR.

Note: The RGM_FERD register can be written only once between two power-on reset events.

An enabled external reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM_FESS register enables the further configuring of the reset sequence triggered by the external reset. When RGM_FESS.SS_EXR is set, the external reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially when an external reset should not reset the flash.

The MC_RGM may also assert the external reset if the reset sequence was triggered by one of the following:

- a power-on reset
- a ‘destructive’ reset event
- an external reset event
- a ‘functional’ reset event configured via the RGM_FBRE register to assert the external reset

In this case, the external reset is asserted until the end of PHASE3.

8.4.4 Functional Resets

A ‘functional’ reset indicates that an event has occurred after which it can be guaranteed that critical register and memory content is still intact.

The status flag associated with a given ‘functional’ reset event (RGM_FES.F_<functional reset> bit) is set when the ‘functional’ reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously,

and it is software's responsibility to determine which reset source is the most critical for the application.

The 'functional' reset can be optionally disabled by software writing bit RGM_FERD.D_<functional reset>.

Note: The RGM_FERD register can be written only once between two power-on reset events.

An enabled 'functional' reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM_FESS register enables the further configuring of the reset sequence triggered by a functional reset. When RGM_FESS.SS_<functional reset> is set, the associated 'functional' reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially in case a functional reset should not reset the flash module.

See the MC_ME chapter for details on the STANDBY0 and DRUN modes.

8.4.5 Alternate Event Generation

The MC_RGM provides alternative events to be generated on reset source assertion. When a reset source is asserted, the MC_RGM normally enters the reset sequence. Alternatively, it is possible for some reset source events to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the RGM_FERD and RGM_FEAR registers as shown in *Table 75*.

Table 75. MC_RGM Alternate Event Selection

RGM_FERD Bit Value	RGM_FEAR Bit Value	Generated Event
0	X	reset
1	0	SAFE mode request
1	1	interrupt request

The alternate event is cleared by deasserting the source of the request (i.e., at the reset source that caused the alternate request) and also clearing the appropriate RGM_FES status bit.

Note: Alternate requests (SAFE mode as well as interrupt requests) are generated regardless of whether the system clock is running.

Note: If a masked 'functional' reset event which is configured to generate a SAFE mode/interrupt request occurs during PHASE1, it is ignored, and the MC_RGM will not send any safe mode/interrupt request to the MC_ME.

8.4.6 Boot Mode Capturing

The MC_RGM samples PAD[4:2] whenever $\overline{\text{RESET}}$ is asserted until five internal RC oscillator clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after $\overline{\text{RESET}}$ has been deasserted for subsequent boots after reset sequences during which $\overline{\text{RESET}}$ is not asserted.

Note: In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that $\overline{RESET}$ is asserted.

Note: $\overline{RESET}$ can be asserted as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See Table 74 for details.)

9 Interrupt Controller (INTC)

9.1 Introduction

The INTC provides priority-based preemptive scheduling of interrupt service requests (ISRs). This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 148 interrupt requests. It is targeted to work with Power Architecture technology and automotive applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.The SPC56xP60/54 implements INTC_0 and INTC_1

For high-priority interrupt requests in these target applications, the time from the assertion of the peripheral's interrupt request to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Because each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that tasks sharing the resource will not preempt each other.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests. These software configurable interrupt requests can also be used to separate the work involved in servicing an interrupt request into a high-priority portion and a low-priority portion. The high-priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software configurable interrupt request to finish the servicing in a lower priority ISR. Therefore these software configurable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

9.2 Features

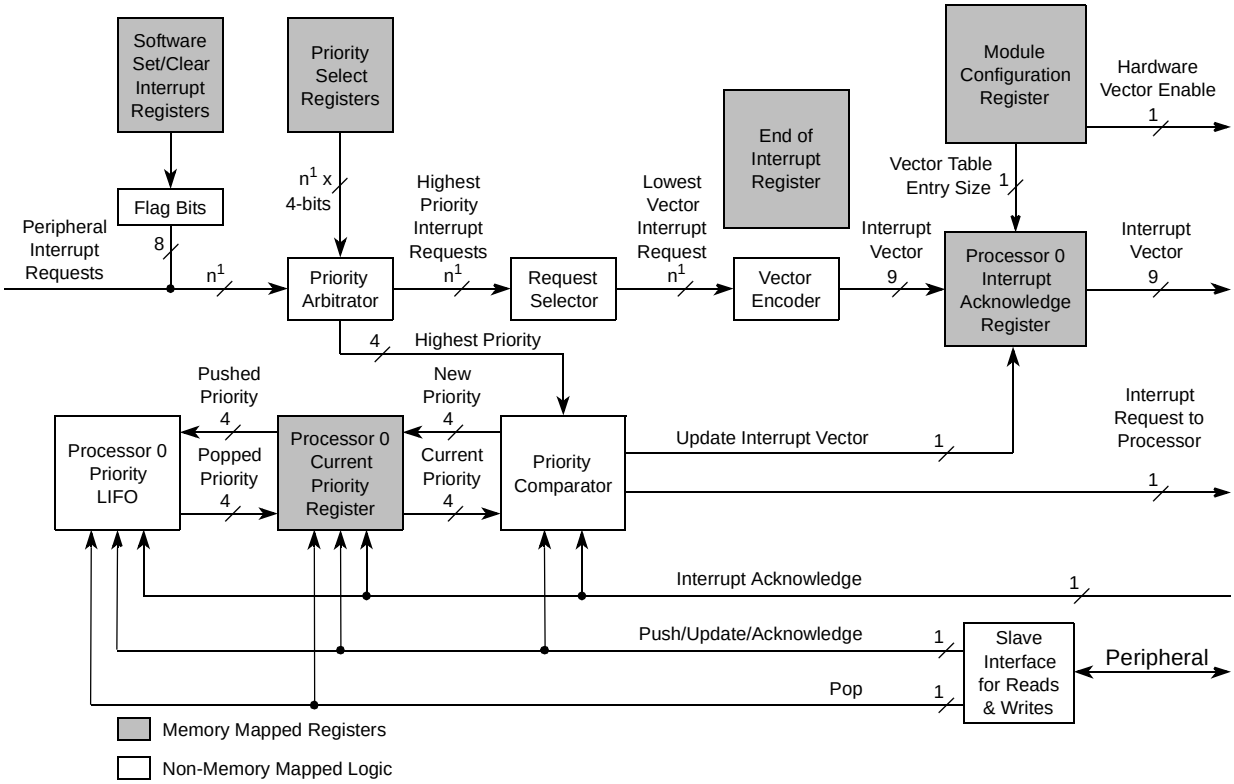
- Supports 140 peripheral interrupt and 8 software-configurable interrupt request sources
- Unique 9-bit vector per interrupt source
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts ISRs or tasks at lower priorities
  - Automatic pushing or popping of preempted priority to or from a LIFO
  - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency—3 clock cycles from receipt of interrupt request from peripheral to interrupt request to processor

Table 76. Interrupt sources available

Interrupt sources (148)	Number available
Software	8
ECSM	3
eDMA2x	17
SWT	1
STM	4
SIUL	4
MC_ME	4
MC_RGM	1
XOSC	1
PIT	4
ADC	2
FlexCAN	16
FlexRay	8
eTimer	15
CTU	15
Safety Port	8
DSPI	25
LINFlex	6
FCCU	4
Semaphore	2

9.3 Block diagram

Figure 84 shows a block diagram of the interrupt controller (INTC).



1. The total number of interrupt sources is 148, which includes 16 reserved sources and 8 software sources.

Figure 84. INTC block diagram

9.4 Modes of operation

9.4.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

Note: To correctly configure the interrupts in both software and hardware vector mode, the user must also configure the IVPR. The core register IVPR contains the base address for the interrupt handlers. Please refer to the core reference manual for more information.

Software vector mode

In software vector mode, the interrupt exception handler software must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated HVEN bit in INTC_MCR is negated. The hardware vector enable signal to processor 0 or processor 1 is driven as negated when its associated HVEN bit is negated. The vector is read from

INC_IACKR. Reading the INTC_IACKR negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in INTC_CPR onto the associated LIFO and updates PRI in the associated INTC_CPR with the new priority.

Furthermore, the interrupt vector to the processor is driven as all 0s. The interrupt acknowledge signal from the associated processor is ignored.

Hardware vector mode

In hardware vector mode, the hardware signals the interrupt vector from the INTC in conjunction with a processor that can use that vector. This hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore, the interrupt exception handler is specific to a peripheral or software configurable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when the associated HVEN bit in the INTC_MCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software configurable interrupt request. The vector value matches the value of the INTVEC field in the INTC_IACKR field in the INTC_IACKR, depending on which processor was assigned to handle a given interrupt source.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for the interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC_CPR register onto the associated LIFO and updates the associated PRI in the associated INTC_CPR register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC_CPR does not occur when the associated interrupt acknowledge signal asserts and INTC_SSCIR0_3–INTC_SSCIR4_7 is written at a time such that the PRI value in the associated INTC_CPR register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the associated INTC_CPR is updated with the new priority, and the associated LIFO is neither pushed or popped.

Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

Stop mode

The INTC supports stop mode. The INTC can have its clock input disabled at any time by the clock driver on the device. While its clocks are disabled, the INTC registers are not accessible.

The INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor.

9.5 Memory map and registers description

9.5.1 Module memory map

Table 77 shows the INTC memory map.

Table 77. INTC memory map

Offset from INTC_BASE 0xFFF4_8000 (INTC_0) 0x8FF4_8000 (INTC_1)	Register	Location
0x0000	*INTC Module Configuration Register (INTC_MCR)*	*on page 233*
0x0004	Reserved	
0x0008	*INTC Current Priority Register (INTC_CPR)*	*on page 233*
0x000C	Reserved	
0x0010	*INTC Interrupt Acknowledge Register(INTC_IACKR)*	*on page 235*
0x0014	Reserved	
0x0018	*INTC End-of-Interrupt Register (INTC_EOIR)*	*on page 236*
0x001C	Reserved	
0x0020–0x0027	*INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)*	*on page 237*
0x0028– 0x003C	Reserved	
0x0040–0x0144	*INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260)*(1)	*on page 238*
0x0148–0x3FFF	Reserved	

1. The PRI fields are “reserved” for peripheral interrupt requests whose vectors are labeled as Reserved in *Table 84*.

9.5.2 Registers description

With exception of the INTC_SSCI*n* and INTC_PSR*n*, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, misaligned 16 bits to the middle 2 bytes, and aligned 32 bits.

Although INTC_SSCI*n* and INTC_PSR*n* are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of INTC_IACKR are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC_SSCIR0_3–INTC_SSCIR4_7 or INTC_EOIR does not affect the operation of the write.

INTC registers are accessible only when the core is in supervisor mode.

INTC Module Configuration Register (INTC_MCR)

The module configuration register configures options of the INTC.

Figure 85. INTC Module Configuration Register (INTC_MCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	VTES	0	0	0	0	HVEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 78. INTC_MCR field descriptions

Field	Description
26 VTES	Vector table entry size Controls the number of 0s to the right of INTVEC in *Section : INTC Interrupt Acknowledge Register(INTC_IACKR).* If the contents of INTC_IACKR are used as an address of an entry in a vector table as in software vector mode, then the number of right most 0s will determine the size of each vector table entry. VTES impacts software vector mode operation but also affects INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode. 0 4 bytes 1 8 bytes
31 HVEN	Hardware vector enable Controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 9.4: Modes of operation,* for the details of the handshaking with the processor in each mode. 0 Software vector mode 1 Hardware vector mode

INTC Current Priority Register (INTC_CPR)

Figure 86. INTC Current Priority Register (INTC_CPR)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Table 79. INTC_CPR field descriptions

Field	Description
28–31 PRI[0:3]	Priority PRI is the priority of the currently executing ISR according to the following: 1111 Priority 15—highest priority 1110 Priority 14 1101 Priority 13 1100 Priority 12 1011 Priority 11 1010 Priority 10 1001 Priority 9 1000 Priority 8 0111 Priority 7 0110 Priority 6 0101 Priority 5 0100 Priority 4 0011 Priority 3 0010 Priority 2 0001 Priority 1 0000 Priority 0—lowest priority

The INTC_CPR masks any peripheral or software settable interrupt request set at the same or lower priority as the current value of the INTC_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC_IACKR) is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC end-of-interrupt register (INTC_EOIR) is written, the LIFO is popped into the INTC_CPR's PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to *Section 9.7.5: Priority ceiling protocol.*

Note: A store to modify the PRI field that closely precedes or follows an access to a shared resource can result in a non-coherent access to the resource. Refer to Section : Ensuring coherency, for example code to ensure coherency.

INTC Interrupt Acknowledge Register(INTC_IACKR)

Figure 87. INTC Interrupt Acknowledge Register (INTC_IACKR)

Address Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA (most significant 16 bits)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA (least significant 5 bits)					INTVEC[1]									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. When the VTES bit in INTC_MCR is asserted, INTVEC is shifted to the left one bit. Bit 29 is read as a ‘0’. VTBA is narrowed to 20 bits in width.

Table 80. INTC_IACKR field descriptions

Field	Description
0–20 or 0–19 VTBA	Vector Table Base Address Can be the base address of a vector table of addresses of ISRs. The VTBA only uses the leftmost 20 bits when the VTES bit in INTC_MCR is asserted.
21–29 or 20–28 INTVEC	Interrupt Vector It is the vector of the peripheral or software configurable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode. **Note:** If INTC_MCR[VTES] = 1, then the INTVEC field is shifted left one position to bits 20–28. VTBA is then shortened by one bit to bits 0–19.

The interrupt acknowledge register provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, the INTC_IACKR has side effects from reads. Therefore, it must not be speculatively read while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR does not have side effects in hardware vector mode.

INTC End-of-Interrupt Register (INTC_EOIR)

Figure 88. INTC Interrupt Acknowledge Register 1 (INTC_IACKR1)

Address Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VTBA (most significant 16 bits)															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA (least significant 5 bits)					INTVEC[1]									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. When the VTES bit in INTC_MCR is asserted, INTVEC is shifted to the left one bit. Bit 29 is read as a '0'. VTBA is narrowed to 20 bits in width.

Figure 89. INTC End-of-Interrupt Register (INTC_EOIR)

Address Base + 0x0018 Access: Write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR is written, the priority last pushed on the LIFO is popped into INTC_CPR. An exception to this behavior is described in *Section : Hardware vector mode.* The values and size of data written to the INTC_EOIR are ignored. The values and sizes written to this register neither update the INTC_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR.

Reading the INTC_EOIR has no effect on the LIFO.

INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–

Figure 90. INTC End-of-Interrupt Register 1 (INTC_EOIR1)

Address Base + 0x0018 Access: Write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INTC_SSCIR4_7)

Figure 91. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

Address Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR 0	0	0	0	0	0	0	0	CLR 1
W							SET0								SET1	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR 2	0	0	0	0	0	0	0	CLR 3
W							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 92. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

Address Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR 4	0	0	0	0	0	0	0	CLR 5
W							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR 6	0	0	0	0	0	0	0	CLR 7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 81. INTC_SSCIR[0:7] field descriptions

Field	Description
6, 14, 22, 30 SET[0:7]	Set Flag Bits Writing a ‘1’ sets the corresponding CLR*x* bit. Writing a ‘0’ has no effect. Each SET*x* always will be read as a ‘0’.
7, 15, 23, 31 CLR[0:7]	Clear Flag Bits CLR*x* is the flag bit. Writing a ‘1’ to CLR*x* clears it provided that a ‘1’ is not written simultaneously to its corresponding SET*x* bit. Writing a ‘0’ to CLR*x* has no effect. 0 Interrupt request not pending within INTC 1 Interrupt request pending within INTC

The software set/clear interrupt registers support the setting or clearing of software configurable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. Excepting being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a ‘1’ to SET*x* will leave SET*x* unchanged at 0 but sets CLR*x*. Writing a ‘0’ to SET*x* has no effect. CLR*x* is the flag bit. Writing a ‘1’ to CLR*x* clears it. Writing a ‘0’ to CLR*x* has no effect. If a ‘1’ is written simultaneously to a pair of SET*x* and CLR*x* bits, CLR*x* will be asserted, regardless of whether CLR*x* was asserted before the write.

INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260)

Figure 93. INTC Priority Select Register 0–3 (INTC_PSR[0:3])

Address Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI0				0	0	0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI2				0	0	0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 94. INTC Priority Select Register 260 (INTC_PSR[260])

Address Base + 0x0144 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI260				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 82. INTC_PSR0_3–INTC_PSR260 field descriptions

Field	Description
4–7, 12–15, 20–23, 28–31 PRI[0:3]–PRI260	Priority Select PRI*x* selects the priority for interrupt requests. Refer to *Section 9.6: Functional description.*

Table 83. INTC Priority Select Register address offsets

INTC_PSR*x*_*x*	Offset Address	INTC_PSR*x*_*x*	Offset Address
INTC_PSR0_3	0x0040	INTC_PSR132_135	0x00C4
INTC_PSR4_7	0x0044	INTC_PSR136_139	0x00C8
INTC_PSR8_11	0x0048	INTC_PSR140_143	0x00CC
INTC_PSR12_15	0x004C	INTC_PSR144_147	0x00D0
INTC_PSR16_19	0x0050	INTC_PSR148_151	0x00D4
INTC_PSR20_23	0x0054	INTC_PSR152_155	0x00D8
INTC_PSR24_27	0x0058	INTC_PSR156_159	0x00DC
INTC_PSR28_31	0x005C	INTC_PSR160_163	0x00E0
INTC_PSR32_35	0x0060	INTC_PSR164_167	0x00E4
INTC_PSR36_39	0x0064	INTC_PSR168_171	0x00E8
INTC_PSR40_43	0x0068	INTC_PSR172_175	0x00EC
INTC_PSR44_47	0x006C	INTC_PSR176_179	0x00F0
INTC_PSR48_51	0x0070	INTC_PSR180_183	0x00F4
INTC_PSR52_55	0x0074	INTC_PSR184_187	0x00F8
INTC_PSR56_59	0x0078	INTC_PSR188_191	0x00FC
INTC_PSR60_63	0x007C	INTC_PSR192_195	0x0100
INTC_PSR64_67	0x0080	INTC_PSR196_199	0x0104
INTC_PSR68_71	0x0084	INTC_PSR200_203	0x0108

Table 83. INTC Priority Select Register address offsets (continued)

INTC_PSR*x*_*x*	Offset Address	INTC_PSR*x*_*x*	Offset Address
INTC_PSR72_75	0x0088	INTC_PSR204_207	0x010C
INTC_PSR76_79	0x008C	INTC_PSR208_211	0x0110
INTC_PSR80_83	0x0090	INTC_PSR212_215	0x0114
INTC_PSR84_87	0x0094	INTC_PSR216_219	0x0118
INTC_PSR88_91	0x0098	INTC_PSR220_223	0x011C
INTC_PSR92_95	0x009C	INTC_PSR224_227	0x0120
INTC_PSR96_99	0x00A0	INTC_PSR228_231	0x0124
INTC_PSR100_103	0x00A4	INTC_PSR232_235	0x0128
INTC_PSR104_107	0x00A8	INTC_PSR236_239	0x012C
INTC_PSR108_111	0x00AC	INTC_PSR240_243	0x0130
INTC_PSR112_115	0x00B0	INTC_PSR244_247	0x0134
INTC_PSR116_119	0x00B4	INTC_PSR248_251	0x0138
INTC_PSR120_123	0x00B8	INTC_PSR252_255	0x013C
INTC_PSR124_127	0x00BC	INTC_PSR256_259	0x0140
INTC_PSR128_131	0x00C0	INTC_PSR260	0x0144

9.6 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

Note: The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose PRIn value in INTC_PSR0–INTC_PSR260 is higher than the PRI value in INTC_CPR, negates before the interrupt request to the processor for that peripheral or software settable interrupt request is acknowledged, the interrupt request to the processor still can assert or will remain asserted for that peripheral or software settable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software settable interrupt request. Also, the PRI value in the INTC_CPR will be updated with the corresponding PRIn value in INTC_PSRn. Furthermore, clearing the peripheral interrupt request's enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

Table 84. Interrupt vector table

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
On-Platform Peripherals				
Software Interrupts				
0	0x0800	Software setable flag 0	INTC_0 (Software)	INTC_1 (Software)
1	0x0804	Software setable flag 1	INTC_0 (Software)	INTC_1 (Software)
2	0x0808	Software setable flag 2	INTC_0 (Software)	INTC_1 (Software)
3	0x080C	Software setable flag 3	INTC_0 (Software)	INTC_1 (Software)
4	0x0810	Software setable flag 4	INTC_0 (Software)	INTC_1 (Software)
5	0x0814	Software setable flag 5	INTC_0 (Software)	INTC_1 (Software)
6	0x0818	Software setable flag 6	INTC_0 (Software)	INTC_1 (Software)
7	0x081C	Software setable flag 7	INTC_0 (Software)	INTC_1 (Software)
8	0x0820		Reserved	
ECSM				
9	0x0824	Platform Flash Bank 0 Abort | Platform Flash Bank 0 Stall | Platform Flash Bank 1 Abort | Platform Flash Bank 1 Stall | Platform Flash Bank 2 Abort | Platform Flash Bank 2 Stall | Platform Flash Bank 3 Abort | Platform Flash Bank 3 Stall	ECSM_0	ECSM_1
DMA2x				
10	0x0828	Combined Error	DMA_0	

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
11	0x082C	Channel 0	DMA_0	
12	0x0830	Channel 1	DMA_0	
13	0x0834	Channel 2	DMA_0	
14	0x0838	Channel 3	DMA_0	
15	0x083C	Channel 4	DMA_0	
16	0x0840	Channel 5	DMA_0	
17	0x0844	Channel 6	DMA_0	
18	0x0848	Channel 7	DMA_0	
19	0x084C	Channel 8	DMA_0	
20	0x0850	Channel 9	DMA_0	
21	0x0854	Channel 10	DMA_0	
22	0x0858	Channel 11	DMA_0	
23	0x085C	Channel 12	DMA_0	
24	0x0860	Channel 13	DMA_0	
25	0x0864	Channel 14	DMA_0	
26	0x0868	Channel 15	DMA_0	
27	0x086C	Reserved		
SWT				
28	0x0870	SWT Timeout	SWT_0	SWT_1
29	0x0874	Reserved		
STM				
30	0x0878	Match on channel 0	STM_0	STM_1
31	0x087C	Match on channel 1	STM_0	STM_1
32	0x0880	Match on channel 2	STM_0	STM_1
33	0x0884	Match on channel 3	STM_0	STM_1
34	0x0888	Reserved		
ECSM				
35	0x088C	ECC_DBD_PlatformFlash | ECC_DBD_PlatformRAM	ECSM_0	ECSM_1
36	0x0890	ECC_SBC_PlatformFlash | ECC_SBC_PlatformRAM	ECSM_0	ECSM_1
37	0x0894	Reserved		
38	0x0898	Reserved		
39	0x089C	Reserved		

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
40	0x08A0	Reserved		
SIUL				
41	0x08A4	SIU External IRQ_0	SIUL	
42	0x08A8	SIU External IRQ_1	SIUL	
43	0x08AC	SIU External IRQ_2	SIUL	
44	0x08B0	SIU External IRQ_3	SIUL	
45	0x08B4	Reserved		
46	0x08B8	Reserved		
47	0x08BC	Reserved		
48	0x08C0	Reserved		
49	0x08C4	Reserved		
50	0x08C8	Reserved		
MC_ME				
51	0x08CC	Safe Mode Interrupt	MC_ME	
52	0x08D0	Mode Transition Interrupt	MC_ME	
53	0x08D4	Invalid Mode Interrupt	MC_ME	
54	0x08D8	Invalid Mode Config	MC_ME	
55	0x08DC	Reserved		
MC_RGM				
56	0x08E0	Functional and destructive reset alternate event interrupt (ipi_int)	MC_RGM	
XOSC				
57	0x08E4	XOSC counter expired (ipi_int_osc)	XOSC	
PIT				
58	0x08E8	Reserved		
59	0x08EC	PITimer Channel 0	PIT	
60	0x08F0	PITimer Channel 1	PIT	
61	0x08F4	PITimer Channel 2	PIT	
ADC0				
62	0x08F8	ADC_EOC	ADC_0	
63	0x08FC	Reserved		
64	0x0900	ADC_WD	ADC_0	

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
FlexCAN0				
65	0x0904	FLEXCAN_ESR[ERR_INT]	FlexCAN_0	
66	0x0908	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_0	
67	0x090C	FLEXCAN_ESR_WAK	FlexCAN_0	
68	0x0910	FLEXCAN_BUF_00_03	FlexCAN_0	
69	0x0914	FLEXCAN_BUF_04_07	FlexCAN_0	
70	0x0918	FLEXCAN_BUF_08_11	FlexCAN_0	
71	0x091C	FLEXCAN_BUF_12_15	FlexCAN_0	
72	0x0920	FLEXCAN_BUF_16_31	FlexCAN_0	
73	0x0924	Reserved		
DSPI0				
74	0x0928	DSPI_SR[TFUF] / DSPI_SR[RFOF]	DSPI_0	
75	0x092C	DSPI_SR[EOQF]	DSPI_0	
76	0x0930	DSPI_SR[TFFF]	DSPI_0	
77	0x0934	DSPI_SR[TCF]	DSPI_0	
78	0x0938	DSPI_SR[RFDF]	DSPI_0	
LINFlex0				
79	0x093C	LINFlex_RXI	LINFlex_0	
80	0x0940	LINFlex_TXI	LINFlex_0	
81	0x0944	LINFlex_ERR	LINFlex_0	
82	0x0948	Reserved		
83	0x094C	Reserved		
84	0x0950	Reserved		
FlexCAN1				
85	0x0954	FLEXCAN_ESR[ERR_INT]	FlexCAN_1	
86	0x0958	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_1	
87	0x095C	FLEXCAN_ESR_WAK	FlexCAN_1	
88	0x0960	FLEXCAN_BUF_00_03	FlexCAN_1	
89	0x0964	FLEXCAN_BUF_04_07	FlexCAN_1	

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
90	0x0968	FLEXCAN_BUF_08_11	FlexCAN_1	
91	0x096C	FLEXCAN_BUF_12_15	FlexCAN_1	
92	0x0970	FLEXCAN_BUF_16_31	FlexCAN_1	
93	0x0974	Reserved		
DSPI1				
94	0x0978	DSPI_SR[TFUF] / DSPI_SR[RFOF]	DSPI_1	
95	0x097C	DSPI_SR[EOQF]	DSPI_1	
96	0x0980	DSPI_SR[TFFF]	DSPI_1	
97	0x0984	DSPI_SR[TCF]	DSPI_1	
98	0x0988	DSPI_SR[RFDF]	DSPI_1	
LINFlex1				
99	0x098C	LINFlex_RXI	LINFlex_1	
100	0x0990	LINFlex_TXI	LINFlex_1	
101	0x0994	LINFlex_ERR	LINFlex_1	
102	0x0998	Reserved		
103	0x099C	Reserved		
104	0x09A0	Reserved		
105	0x09A4	Reserved		
106	0x09A8	Reserved		
107	0x09AC	Reserved		
108	0x09B0	Reserved		
109	0x09B4	Reserved		
110	0x09B8	Reserved		
111	0x09BC	Reserved		
112	0x09C0	Reserved		
113	0x09C4	Reserved		
DSPI2				
114	0x09C8	DSPI_SR[TFUF] / DSPI_SR[RFOF]	DSPI_2	
115	0x09CC	DSPI_SR[EOQF]	DSPI_2	
116	0x09D0	DSPI_SR[TFFF]	DSPI_2	
117	0x09D4	DSPI_SR[TCF]	DSPI_2	
118	0x09D8	DSPI_SR[RFDF]	DSPI_2	

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
119	0x09DC	Reserved		
120	0x09E0	Reserved		
121	0x09E4	Reserved		
122	0x09E8	Reserved		
123	0x09EC	Reserved		
124	0x09F0	Reserved		
125	0x09F4	Reserved		
126	0x09F8	Reserved		
PIT				
127	0x09FC	PITimer Channel 3	PIT	
128	0x0A00	Reserved		
129	0x0A04	Reserved		
130	0x0A08	Reserved		
131	0x0A0C	Reserved		
132	0x0A10	Reserved		
FlexRay				
133	0x0A14	CIFRR.FNEAIF	FlexRay	
134	0x0A18	CIFRR.FNEBIF	FlexRay	
135	0x0A1C	CIFRR.WUPIF	FlexRay	
136	0x0A20	CIFRR.PRIF	FlexRay	
137	0x0A24	CIFRR.CHIF	FlexRay	
138	0x0A28	CIFRR.TBIF	FlexRay	
139	0x0A2C	CIFRR.RBIF	FlexRay	
140	0x0A30	CIFRR.MIF	FlexRay	
141	0x0A34	Reserved		
142	0x0A38	Reserved		
143	0x0A3C	Reserved		
144	0x0A40	Reserved		
145	0x0A44	Reserved		
146	0x0A48	Reserved		
147	0x0A4C	Reserved		
148	0x0A50	Reserved		
149	0x0A54	Reserved		
150	0x0A58	Reserved		

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
151	0x0A5C	Reserved		
152	0x0A60	Reserved		
153	0x0A64	Reserved		
154	0x0A68	Reserved		
155	0x0A6C	Reserved		
156	0x0A70	Reserved		
eTimer				
157	0x0A74	TC0IR	eTimer_0	
158	0x0A78	TC1IR	eTimer_0	
159	0x0A7C	TC2IR	eTimer_0	
160	0x0A80	TC3IR	eTimer_0	
161	0x0A84	TC4IR	eTimer_0	
162	0x0A88	TC5IR	eTimer_0	
163	0x0A8C	Reserved		
164	0x0A90	Reserved		
165	0x0A94	WTIF	eTimer_0	
166	0x0A98	Reserved		
167	0x0A9C	RCF	eTimer_0	
168	0x0AA0	TC0IR	eTimer_1	
169	0x0AA4	TC1IR	eTimer_1	
170	0x0AA8	TC2IR	eTimer_1	
171	0x0AAC	TC3IR	eTimer_1	
172	0x0AB0	TC4IR	eTimer_1	
173	0x0AB4	TC5IR	eTimer_1	
174	0x0AB8	Reserved		
175	0x0ABC	Reserved		
176	0x0AC0	Reserved		
177	0x0AC4	Reserved		
178	0x0AC8	RCF	eTimer_1	
179	0x0ACC	Reserved		
180	0x0AD0	Reserved		
181	0x0AD4	Reserved		
182	0x0AD8	Reserved		
183	0x0ADC	Reserved		

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
184	0x0AE0	Reserved		
185	0x0AE4	Reserved		
186	0x0AE8	Reserved		
187	0x0AEC	Reserved		
188	0x0AF0	Reserved		
189	0x0AF4	Reserved		
190	0x0AF8	Reserved		
191	0x0AFC	Reserved		
192	0x0B00	Reserved		
CTU				
193	0x0B04	MRS_I	CTU_0	
194	0x0B08	T0_I	CTU_0	
195	0x0B0C	T1_I	CTU_0	
196	0x0B10	T2_I	CTU_0	
197	0x0B14	T3_I	CTU_0	
198	0x0B18	T4_I	CTU_0	
199	0x0B1C	T5_I	CTU_0	
200	0x0B20	T6_I	CTU_0	
201	0x0B24	T7_I	CTU_0	
202	0x0B28	FIFO1_I	CTU_0	
203	0x0B2C	FIFO2_I	CTU_0	
204	0x0B30	FIFO3_I	CTU_0	
205	0x0B34	FIFO4_I	CTU_0	
206	0x0B38	ADC_I	CTU_0	
207	0x0B3C	ERR_I	CTU_0	
SafetyPort				
208	0x0B40	FLEXCAN_ESR[ERR_INT]	SafetyPort (FlexCAN)	
209	0x0B44	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	SafetyPort (FlexCAN)	
210	0x0B48	FLEXCAN_ESR_WAK	SafetyPort (FlexCAN)	
211	0x0B4C	FLEXCAN_BUF_0_3	SafetyPort (FlexCAN)	
212	0x0B50	FLEXCAN_BUF_4_7	SafetyPort (FlexCAN)	
213	0x0B54	FLEXCAN_BUF_8_11	SafetyPort (FlexCAN)	

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
214	0x0B58	FLEXCAN_BUF_12_15	SafetyPort (FlexCAN)	
215	0x0B5C	FLEXCAN_BUF_16_31	SafetyPort (FlexCAN)	
216	0x0B60	Reserved		
DSPI3				
217	0x0B64	DSPI_SR[TFUF] / DSPI_SR[RFOF]	DSPI_3	
218	0x0B68	DSPI_SR[EOQF]	DSPI_3	
219	0x0B6C	DSPI_SR[TFFF]	DSPI_3	
220	0x0B70	DSPI_SR[TCF]	DSPI_3	
221	0x0B74	DSPI_SR[RFDF]	DSPI_3	
222	0x0B78	Reserved		
223	0x0B7C	Reserved		
224	0x0B80	Reserved		
225	0x0B84	Reserved		
226	0x0B88	Reserved		
227	0x0B8C	Reserved		
228	0x0B90	Reserved		
229	0x0B94	Reserved		
230	0x0B98	Reserved		
231	0x0B9C	Reserved		
232	0x0BA0	Reserved		
233	0x0BA4	Reserved		
234	0x0BA8	Reserved		
235	0x0BAC	Reserved		
236	0x0BB0	Reserved		
237	0x0BB4	Reserved		
238	0x0BB8	Reserved		
239	0x0BBC	Reserved		
240	0x0BC0	Reserved		
241	0x0BC4	Reserved		
242	0x0BC8	Reserved		
243	0x0BCC	Reserved		
244	0x0BD0	Reserved		
245	0x0BD4	Reserved		

Table 84. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Source Module for INTC_0	Source Module for INTC_1
246	0x0BD8	Reserved		
SEM4				
247	0x0BDC	cp0_semaphore_int	Semaphore (SEM4_0)	
		cp1_semaphore_int		Semaphore (SEM4_0)
248	0x0BE0	cp0_semaphore_int	Semaphore (SEM4_1)	
		cp1_semaphore_int		Semaphore (SEM4_1)
249	0x0BE4	Not used	Not used	
FCCU				
250	0x0BE8	irq_alarm_b	FCCU	
251	0x0BEC	irq_misc_b	FCCU	
252	0x0BF0	irq_rccs_b[0]	FCCU	
253	0x0BF4	irq_rccs_b[1]	FCCU	
254	0x0BF8	Reserved		
255	0x0BFC	Reserved		
DSPI4				
256	0x0C00	DSPI_SR[TFUF] / DSPI_SR[RFOF]	DSPI_4	
257	0x0C04	DSPI_SR[EOQF]	DSPI_4	
258	0x0C08	DSPI_SR[TFFF]	DSPI_4	
259	0x0C0C	DSPI_SR[TCF]	DSPI_4	
260	0x0C10	DSPI_SR[RFDF]	DSPI_4	

9.6.1 Interrupt request sources

The INTC has two types of interrupt requests, peripheral and software configurable. These interrupt requests can assert on any clock cycle.

Peripheral interrupt requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

External interrupts are handled by the SIU (see *Section 11.6.4: External interrupts*).

Software configurable interrupt requests

An interrupt request is triggered by software by writing a ‘1’ to a SET*x* bit in *INTC_SSCIR0_3–INTC_SSCIR4_7*. This write sets the corresponding flag bit, CLR*x*, resulting in the interrupt request. The interrupt request is cleared by writing a ‘1’ to the CLR*x* bit.

The time from the write to the SET*x* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

Unique vector for each interrupt request source

Each peripheral and software configurable interrupt request is assigned a hardwired unique 9-bit vector. Software configurable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests. The peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC (see *Table 76*).

9.6.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*x* values set in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260). The result is compared to PRI in the associated *INTC_CPR*. The results of those comparisons manage the priority of the ISR executed by the associated processor. The associated LIFO also assists in managing that priority.

Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in *Figure 84* compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software configurable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software settable interrupt request is generated for INTC interrupt acknowledge register (INTC_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

Priority arbitrator subblock

The priority arbitrator subblock for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software configurable. The output of the priority arbitrator subblock is the highest of those priorities assigned to a given processor. Also, any interrupt requests that have this highest priority are output as asserted interrupt requests to the associated request selector subblock.

Request selector subblock

If only one interrupt request from the associated priority arbitrator subblock is asserted, then it is passed as asserted to the associated vector encoder subblock. If multiple interrupt requests from the associated priority arbitrator subblock are asserted, only the one with the lowest vector passes as asserted to the associated vector encoder subblock. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software configurable interrupt requests.

Vector encoder subblock

The vector encoder subblock generates the unique 9-bit vector for the asserted interrupt request from the request selector subblock for the associated processor.

Priority comparator subblock

The priority comparator submodule compares the highest priority output from the priority arbitrator submodule with PRI in INTC_CPR. If the priority comparator submodule detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in INTC_CPR or the PRI value in INTC_CPR is lowered below this highest priority. This highest priority then becomes the new priority that will be written to PRI in INTC_CPR when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*n* in INTC_PSR*n* are zero will not cause a preemption because their PRI*n* will not be higher than PRI in INTC_CPR.

Last-in first-out (LIFO)

The LIFO stores the preempted PRI values from the INTC_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC_CPR does not need to be loaded from the INTC_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC_CPR.

The PRI value in the INTC_CPR is pushed onto the LIFO when the INTC_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC_CPR whenever the INTC_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop 0s if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

9.6.3 Handshaking with processor

Software vector mode handshaking

This section describes handshaking in software vector mode.

Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode and the handshake near the end of the interrupt exception handler, is shown in *Figure 95*. The INTC examines the peripheral and software configurable interrupt requests. When it finds an asserted peripheral or software configurable interrupt request with a higher priority than PRI in the associated *INTC_CPR,* it asserts the interrupt request to the processor. The INTVEC field in the associated *INTC_IACKR* is updated with the preempting

interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of handshaking process is described in *Section : Software vector mode. "*

End of interrupt exception handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR) must be written.When written, the associated LIFO is popped so the preempted priority is restored into PRI of the INTC_CPR. Before it is written, the peripheral or software configurable flag bit must be cleared so that the peripheral or software configurable interrupt request is negated.

Note: To ensure proper operation across all eSys MCUs, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC_EOIR.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer even be asserted. When PRI in INTC_CPR is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software settable interrupt request at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor will return to the instruction address that it was to next execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

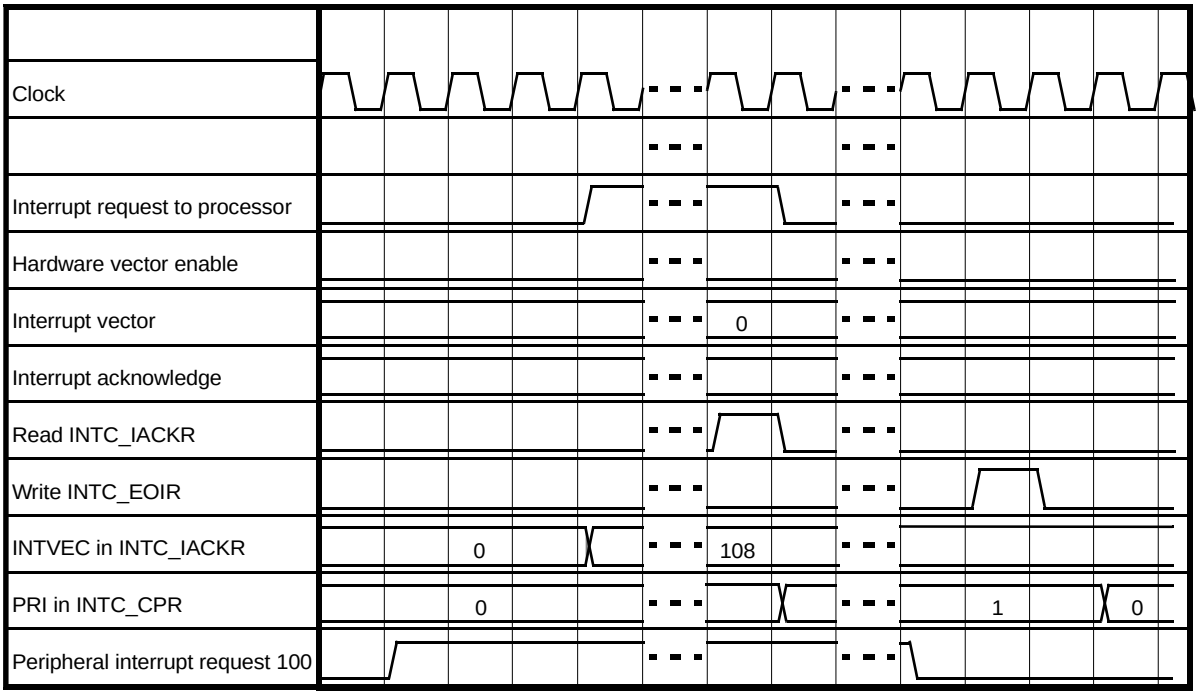


Figure 95. Software vector mode handshaking timing diagram

Hardware vector mode handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in *Figure 96*. As in software vector mode, the INTC examines the peripheral and software settable interrupt requests, and when it finds an asserted one with a higher priority than PRI in INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC_IACKR. The rest of the handshaking is described in" *Section : Hardware vector mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR, is the same as in software vector mode. Refer to *Section : End of interrupt exception handler*.

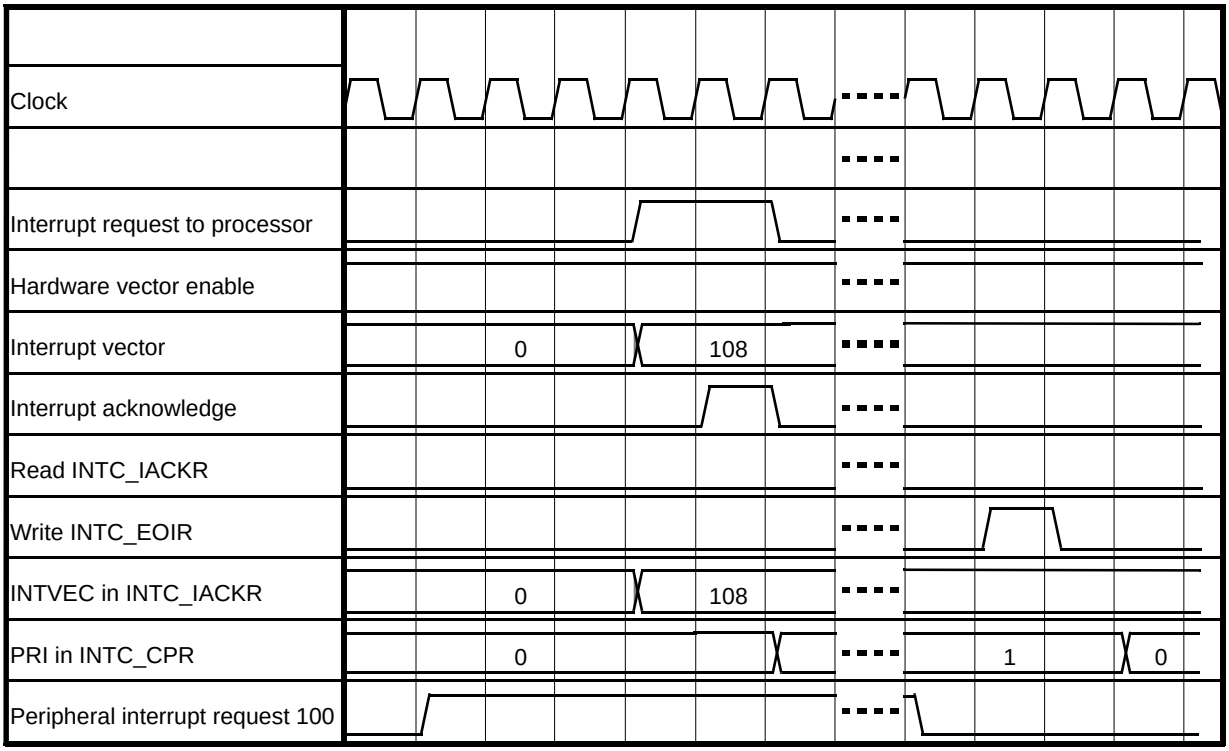


Figure 96. Hardware vector mode handshaking timing diagram

9.7 Initialization/application information

9.7.1 Initialization flow

After exiting reset, all of the PRI*n* fields in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260) will be zero, and PRI in INTC current priority register (INTC_CPR) will be

15. These reset values will prevent the INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is: interrupt_request_initialization:

```
interrupt_request_initialization:
configure VTES and HVEN in INTC_MCR
configure VTBA in INTC_IACKR
raise the PRIn fields in INTC_PSRn
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR to zero
enable processor recognition of interrupts
```

9.7.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

Software vector mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1
lis   r3,INTC_IACKR@ha # form adjusted upper half of INTC_IACKR address
lwz   r3,INTC_IACKR@l(r3) # load INTC_IACKR, which clears request to
processor
lwz   r3,0x0(r3)      # load address of ISR from vector table
wrteei 1              # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr r3               # move INTC_IACKR contents into link register
blrl                  # branch to ISR; link register updated with epilog
                      # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar                  # ensure store to clear flag bit has completed
lis   r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li    r4,0x0          # form 0 to write to INTC_EOIR
wrteei 0              # disable processor recognition of interrupts
stw   r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi


vector_table_base_address:
address of ISR for interrupt with vector 0
```

```
address of ISR for interrupt with vector 1
.
.
.
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRx:
code to service the interrupt event
code to clear flag bit that drives interrupt request to INTC

blr # return to epilog
```

Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. This example assumes that each interrupt_exception_handler*x* only has space for four instructions, and therefore a branch to interrupt_exception_handler_continued*x* is needed.

```
interrupt_exception_handlerx:
b interrupt_exception_handler_continuedx# 4 instructions available, branch
to continue
interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei 1        # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl    ISRx      # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar            # ensure store to clear flag bit has completed
lis   r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li    r4,0x0    # form 0 to write to INTC_EOIR
wrteei 0        # disable processor recognition of interrupts
stw   r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi

ISRx:
code to service the interrupt event
code to clear flag bit that drives interrupt request to INTC
```

```
blr             # branch to epilog
```

9.7.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR) having a value of 0. The RTOS will execute the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR while the shared resource is being accessed.

An ISR whose PRI*n* in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260) has a value of 0 will not cause an interrupt request to the processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit will cause it to remain negated, which consequently also will not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

9.7.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software configurable interrupt requests. However, if multiple peripheral or software configurable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software configurable interrupt requests asserted.

The example in *Table 85* shows the order of execution of both ISRs with different priorities and the same priority

Table 85. Order of ISR execution example

Step #	Step Description	Code Executing at End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108 (1)	ISR208	ISR308	ISR408	Interrupt Exception Handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4

Table 85. Order of ISR execution example (continued)

Step #	Step Description	Code Executing at End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108 (1)	ISR208	ISR308	ISR408	Interrupt Exception Handler	
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software configurable interrupt requests.

9.7.5 Priority ceiling protocol

Elevating priority

The PRI field in *INTC_CPR* is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC_CPR can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For

example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

Ensuring coherency

A scenario can cause non-coherent accesses to the shared resource. For example, ISR1 and ISR2 are both running on the same core and both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing and writes to the INTC_CPR. The instruction following this store is a store to a value in a shared coherent data block. Either immediately before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corruption of a coherent data block, modifications to PRI in INTC_CPR can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

9.7.6 Selecting priorities according to request rates and deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs that have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2, which has a higher priority than ISR3; however, if ISR3 has a deadline of 150 µs, then it has a higher priority than ISR2.

The INTC has 16 priorities, which may be less than the number of ISRs. In this case, the ISRs should be grouped with other ISRs that have similar deadlines. For example, a priority could be allocated for every time the request rate doubles. ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500 µs would share a priority, ISRs with request rates around 250 µs would share a priority, etc. With this approach, a range of ISR request rates of 2^{16} could be included, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, reducing the number of priorities can reduce the size and latency through the interrupt controller. It also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

9.7.7 Software configurable interrupt requests

The software configurable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and they may also be used by processors to interrupt other processors in a multiple processor system.

Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRI*x* value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260), which becomes the PRI value in *INTC_CPR* with the interrupt acknowledge. The ISR, however, can have a portion that does not need to be executed at this higher priority. Therefore, executing the later portion that does not need to be executed at this higher priority can prevent the execution of ISRs that do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*x* bit in *INTC_SSCIR0_3–INTC_SSCIR4_7*. Writing a '1' to SET*x* causes a software configurable interrupt request. This software configurable interrupt request will usually have a lower PRI*x* value in the INTC_PSR*x_x* and will not cause preemptive scheduling inefficiencies. After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

Scheduling an ISR on another processor

Because the SET*x* bits in the INTC_SSCIR*x_x* are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that the processor executing the software configurable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLR*x* bit in INTC_SSCIR*x_x* is asserted before again writing a '1' to the SET*x* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. To do this, the first processor writes a '1' to a SET*x* bit on the second processor. After accessing the block of data, the second processor clears the corresponding CLR*x* bit and then writes 1 to a SET*x* bit on the first processor, informing it that it can now access the block of data.

9.7.8 Lowering priority within an ISR

A common method for avoiding preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (see *Section : Scheduling a lower priority portion of an ISR*) is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

Note: Lowering the PRI value in INTC_CPR within an ISR to below the ISR's corresponding PRI value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260) allows more preemptions than the LIFO depth can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

9.7.9 Negating an interrupt request outside of its ISR

Negating an interrupt request as a side effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and their corresponding interrupt requests. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

Proper setting of interrupt request priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software configurable interrupt requests for these other flag bits must be selected properly. Their PRI*x* values in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR260) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to *INTC_SSCIR0_3–INTC_SSCIR4_7* as the clearing of the flag bit that caused the present ISR to be executed (see *Section : End of interrupt exception handler*).

A flag bit whose enable bit or mask bit negates its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRI*x* value in INTC_PSR*x_x*.

9.7.10 Examining LIFO contents

In normal mode, the user does not need to know the contents of the LIFO. He may not even know how deeply the LIFO is nested. However, if he wants to read the contents, such as in debug mode, they are not memory mapped. The contents can be read by popping the LIFO and reading the PRI field in either *INTC_CPR*. The code sequence is:

```
pop_lifo:
store to INTC_EOIR
load INTC_CPR, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to
pop_lifo
```

When the examination is complete, the LIFO can be restored using this code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR
load INTC_IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

10 System Status and Configuration Module (SSCM)

10.1 Introduction

10.1.1 Overview

The System Status and Configuration Module (SSCM), pictured in *Figure 97*, provides central device functionality.

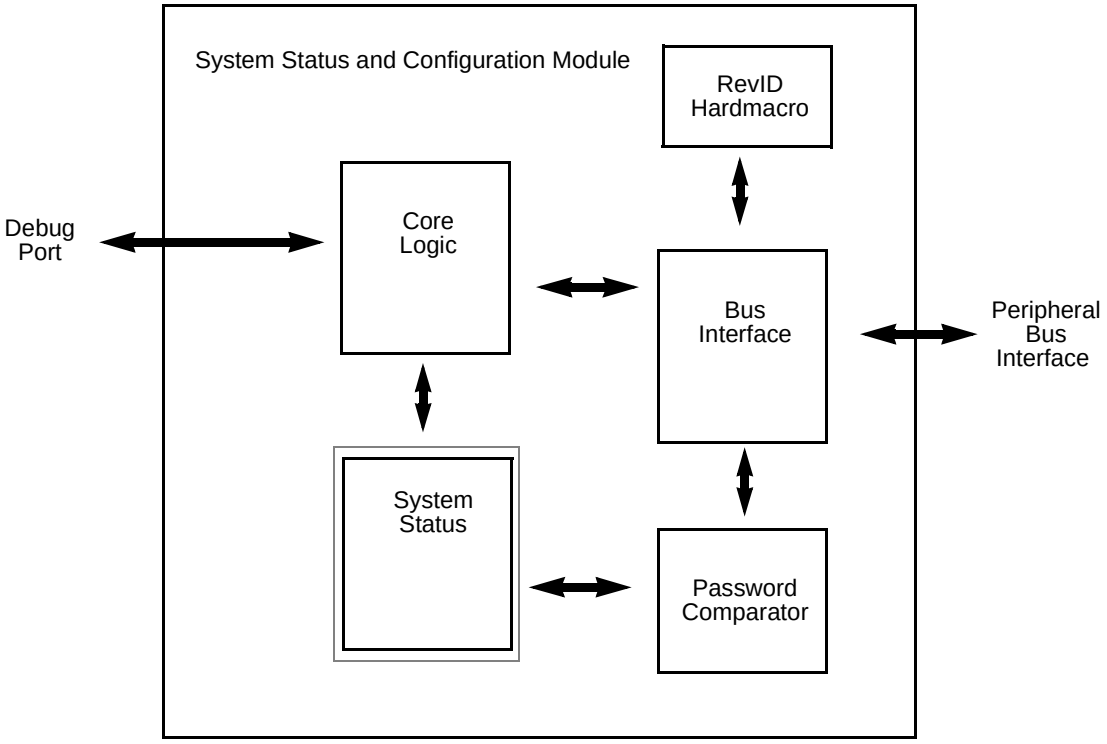


Figure 97. SSCM block diagram

10.1.2 Features

The SSCM includes these features:

- System configuration and status
  - Memory sizes/status
  - Device mode and security status
  - Determine boot vector
  - Search Code Flash for bootable sector
- Device identification information (MCU ID Registers)
- Debug status port enable and selection
- Bus and peripheral abort enable/disable

10.1.3 Modes of operation

The SSCM operates identically in all system modes.

10.2 Memory map and register description

This section provides a detailed description of all memory-mapped registers in the SSCM.

10.2.1 Memory map

Table 86 shows the memory map for the SSCM. Note that all addresses are offsets; the absolute address may be calculated by adding the specified offset to the base address of the SSCM.

Table 86. SSCM memory map

Offset from SSCM_BASE (0xC3FD_8000)	Register	Location
0x0000	STATUS—System Status register	*on page 264*
0x0002	MEMCONFIG—System Memory Configuration register	*on page 265*
0x0004	Reserved (Reads/Writes have no effect)	
0x0006	ERROR—Error Configuration register	*on page 266*
0x0008	DEBUGPORT—Debug Status Port register	*on page 267*
0x000A	Reserved (Reads/Writes have no effect)	
0x000C	PWCMPH—Password Comparison High Word register	*on page 268*
0x0010	PWCMPL—Password Comparison Low Word register	*on page 268*
0x0014	Reserved	
0x0018	DPMBOOT—DPM Boot Register	*on page 270*
0x001C	DPMKEY—DPM Boot Key Register	*on page 270*
0x0020	UOPS—User Option Status Register	*on page 271*
0x0024	Reserved	
0x0028	PSA—Processor Start Address Register	*on page 272*
0x002C–0x3FFF	Reserved	

All registers are accessible via 8, 16 or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the MEMCONFIG register is accessible by a 16-bit read/write to address Base + 0x0002, but performing a 16-bit access to Base + 0x0003 is illegal.

10.2.2 Register description

Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of the registers

is MSB = 0, however the numbering of the internal fields is LSB = 0, for example, register SSCM_STATUS[8] = BMODE[2].

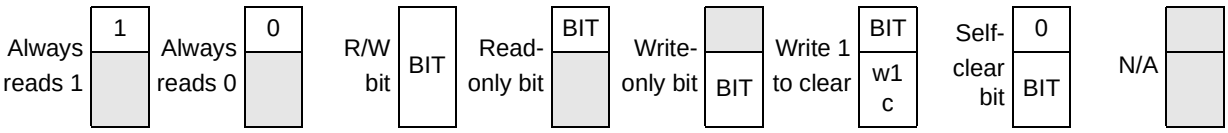


Figure 98. Key to register fields

System Status register (STATUS)

The system status register is a read-only register that reflects the current state of the system.

Figure 99. Status (STATUS) register

Address: Base + 0x0000 Access: Read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	CER	0	NXEN1	NXEN	PUB	SEC	0	BMODE[2:0]			VLE	ABD	0	0	0
W		clear														
RESET:	0	0	0	0	0	0	0	0	0	1	1	0(1)	0	0	0	0

1. The reset value reflects the VLE bit value on RCHW, see *Section 35.5.3: Reset Configuration Half Word (RCHW)*.

Table 87. STATUS allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8, or 0xC).

Table 88. STATUS field descriptions

Field	Description
CER	Configuration Error. This field indicates that the SSCM has detected a configuration error during bootup. 1 Device configuration is not correct 0 No configuration problem detected by the SSCM
NXEN1	Processor 1 Nexus enabled.
NXEN	Nexus enabled
PUB	Public Serial Access Status This bit indicates whether serial boot mode with public password is allowed. 1: Serial boot mode with public password allowed 0: Serial boot mode with private Flash password allowed, provided the key has not been swallowed

Table 88. STATUS field descriptions (continued)

Field	Description
SEC	Security Status This bit reflects the current security state of the Flash. 1: Flash secured 0: Flash not secured
VLE	Variable Length Instruction Mode. When booting from Flash, this field indicates that the code stored there is using the VLE instruction set. The value of this field is determined by the RCHW field of the Flash boot sector. 1 Main Flash contains VLE code 0 Main Flash contains standard PPC code
BMODE [2:0]	Device Boot Mode 000: Reserved for FlexRay boot serial boot loader 001: Legacy bootstrap via CAN (no autobaud) 010: Legacy bootstrap via UART (no autobaud) 011: Single Chip 100–111: Reserved This field is updated only during reset.
ABD	Autobaud Indicates that autobaud detection is active when in SCI or CAN serial boot loader mode. No meaning in other modes.

System Memory Configuration register (MEMCONFIG)

The system memory configuration register is a read-only register that reflects the memory configuration of the system.

Figure 100. System memory configuration (MEMCONFIG) register

Address: Base + 0x0002 Access: Read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	JPIN										IVLD	MREV				DVLD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 89. MEMCONFIG field descriptions

Field	Description
JPIN	JTAG Part ID Number
IVLD	CFlash Valid This bit identifies whether or not the on-chip CFlash is accessible in the system memory map. The Flash may not be accessible due to security limitations. 1: CFlash accessible 0: CFlash not accessible Note: This is a status bit only and writing to this bit does not enable the CFlash if it has been disabled due to specific mode of operation.

Table 89. MEMCONFIG field descriptions (continued)

Field	Description
MREV	Minor Mask Revision
DVLD	DFlash Valid This bit identifies whether or not the on-chip DFlash is visible in the system memory map. The Flash may not be accessible due to security limitations. 1: DFlash visible 0: DFlash not visible Note: This is a status bit only and writing to this bit does not enable the CFlash if it has been disabled due to specific mode of operation.

Table 90. MEMCONFIG allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed (also reads STATUS register)
Write	Not allowed	Not allowed	Not allowed

Error Configuration (ERROR) register

The Error Configuration register is a read-write register that controls the error handling of the system.

Figure 101. Error Configuration (ERROR) register

Address: Base + 0x0006 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAE	RAE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 91. ERROR field descriptions

Field	Description
PAE	Peripheral Bus Abort Enable This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code. 1: Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception. 0: Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception.

Table 91. ERROR field descriptions (continued)

Field	Description
RAE	Register Bus Abort Enable This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code. 1: Illegal accesses to peripherals produce a Prefetch or Data Abort exception. 0: Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception.

Note: Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (i.e., at the PRIDGE level). In this case, the PAE and RAE register bits will have no effect on the abort.

Table 92. ERROR allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed
Write	Allowed	Allowed	Not allowed

Debug Status Port (DEBUGPORT) register

The Debug Status Port register provides debug data on a set of pins.

Figure 102. Debug Status Port (DEBUGPORT) register

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DEBUG_MODE[2:0]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 93. DEBUGPORT field descriptions

Field	Description
13-15 DEBUG_MODE[2:0]	Debug Status Port Mode This field selects the alternate debug functionality for the Debug Status Port. 000: No alternate functionality selected 001: Mode 1 selected 010: Mode 2 selected 011: Mode 3 selected 100: Mode 4 selected 101: Mode 5 selected 110: Mode 6 selected 111: Mode 7 selected *Table 94* describes the functionality of the Debug Status Port in each mode.

Table 94. Debug Status Port modes

Pin (1)	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
0	STATUS[0]	STATUS[8]	MEMCONFIG[0]	MEMCONFIG[8]	Reserved	Reserved	Reserved
1	STATUS[1]	STATUS[9]	MEMCONFIG[1]	MEMCONFIG[9]	Reserved	Reserved	Reserved
2	STATUS[2]	STATUS[10]	MEMCONFIG[2]	MEMCONFIG[10]	Reserved	Reserved	Reserved
3	STATUS[3]	STATUS[11]	MEMCONFIG[3]	MEMCONFIG[11]	Reserved	Reserved	Reserved
4	STATUS[4]	STATUS[12]	MEMCONFIG[4]	MEMCONFIG[12]	Reserved	Reserved	Reserved
5	STATUS[5]	STATUS[13]	MEMCONFIG[5]	MEMCONFIG[13]	Reserved	Reserved	Reserved
6	STATUS[6]	STATUS[14]	MEMCONFIG[6]	MEMCONFIG[14]	Reserved	Reserved	Reserved
7	STATUS[7]	STATUS[15]	MEMCONFIG[7]	MEMCONFIG[15]	Reserved	Reserved	Reserved

1. All signals are active high, unless otherwise noted

Table 95. DEBUGPORT allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Not allowed
Write	Allowed	Allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

Password comparison registers

These registers allow to unsecure the device, if the correct password is known.

Figure 103. Password Comparison Register High Word (PWCMPH) register

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_HI[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_HI[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 104. Password Comparison Register Low Word (PWCMPL) register

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_LO[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_LO[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 96. PWCMPH/L field descriptions

Field	Description
PWD_HI[31:0]	Upper 32 bits of the password
PWD_LO[31:0]	Lower 32 bits of the password

Table 97. PWCMPH/L allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

DPM Boot Register

Figure 105. DPM Boot (DPMBOOT) Register

Address: Base + 0x0018 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	P2BOOT															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	P2BOOT														0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Writes have no effect on this bit

Table 98. DPMBOOT Field Descriptions

Field	Description
P2BOOT	Determines the location from which the 2nd processor will boot, once the main processor releases it from reset. This field is only used if the device is operating in DPM mode.

Boot Key Register

Figure 106. Boot Key Register (DPMKEY) Register

Address: Base + 0x001C Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	KEY															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Writes have no effect on this bit

Table 99. DPMKEY Field Descriptions

Field	Description
KEY	Control key. This field is used to activate the second core in DP Mode. The sequence following sequece is required: - write to the DPMBOOT register - write the value 0101101011110000 (0x5AF0) to the key field - write the value 1010010100001111 (0xA50F) to the key field After this the second core will start executing from the address specified in the DPMBOOT register.

User Option Status Register

Address: Base + 0x0020 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UOPT															
W																
RESET:	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	UOPT															
W																
RESET:	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

= Writes have no effect on this bit

Figure 107. User Option Status (UOPS) Register

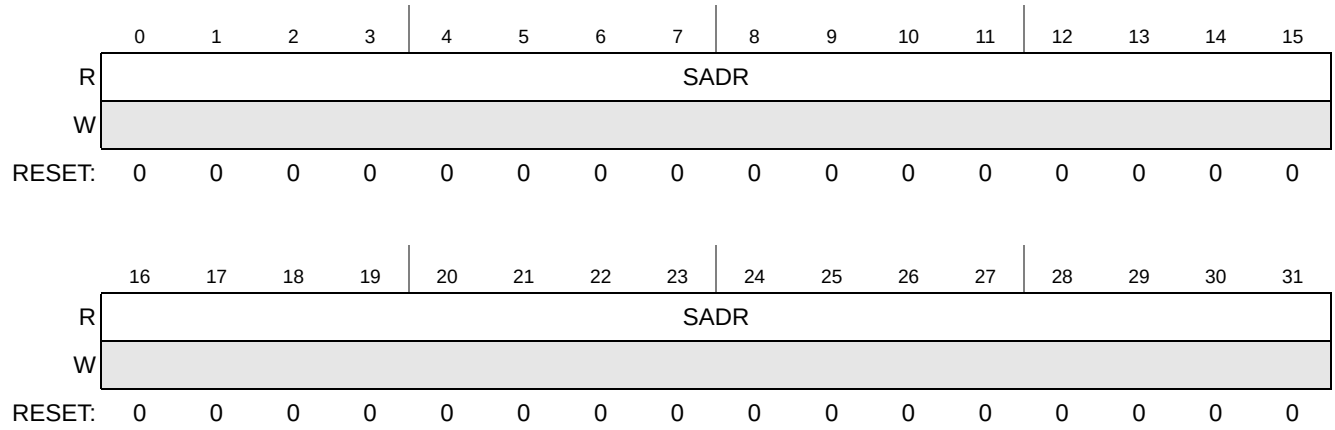
Table 100. UOPS Field Descriptions

Field	Description
UOPT	Shows the values read from the User Option Bits location in the Flash.

Processor Start Address Register

Figure 108. Processor Start Address (PSA) Register

Address: Base + 0x28 Access: Read/Write



= Writes have no effect on this bit

10.3 Functional description

Table 101. PSA Field Descriptions

Field	Description
SADR	Start Address - the boot processor will start executing application code from this address

The primary purpose of the SSCM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debug of the system.

10.4 Initialization/application information

10.4.1 Reset

The reset state of each individual bit is shown in *Section 10.2.2: Register description*.

11 System Integration Unit Lite (SIUL)

11.1 Introduction

This chapter describes the System Integration Unit Lite (SIUL), which is used for the management of the pads and their configuration. It controls the multiplexing of the alternate functions used on all pads and is responsible for managing the external interrupts to the device.

11.2 Overview

The System Integration Unit Lite (SIUL) controls the MCU pad configuration, ports, general-purpose input and output (GPIO) signals and external interrupts with trigger event configuration. *Figure 109* is a block diagram of the SIUL and its interfaces to other system components.

The module provides dedicated general-purpose pads that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the associated output pad. When configured as an input, you can detect the state of the associated pad by reading the value from an internal register. When configured as an input and output, the pad value can be read back, which can be used a method of checking if the written value appeared on the pad.

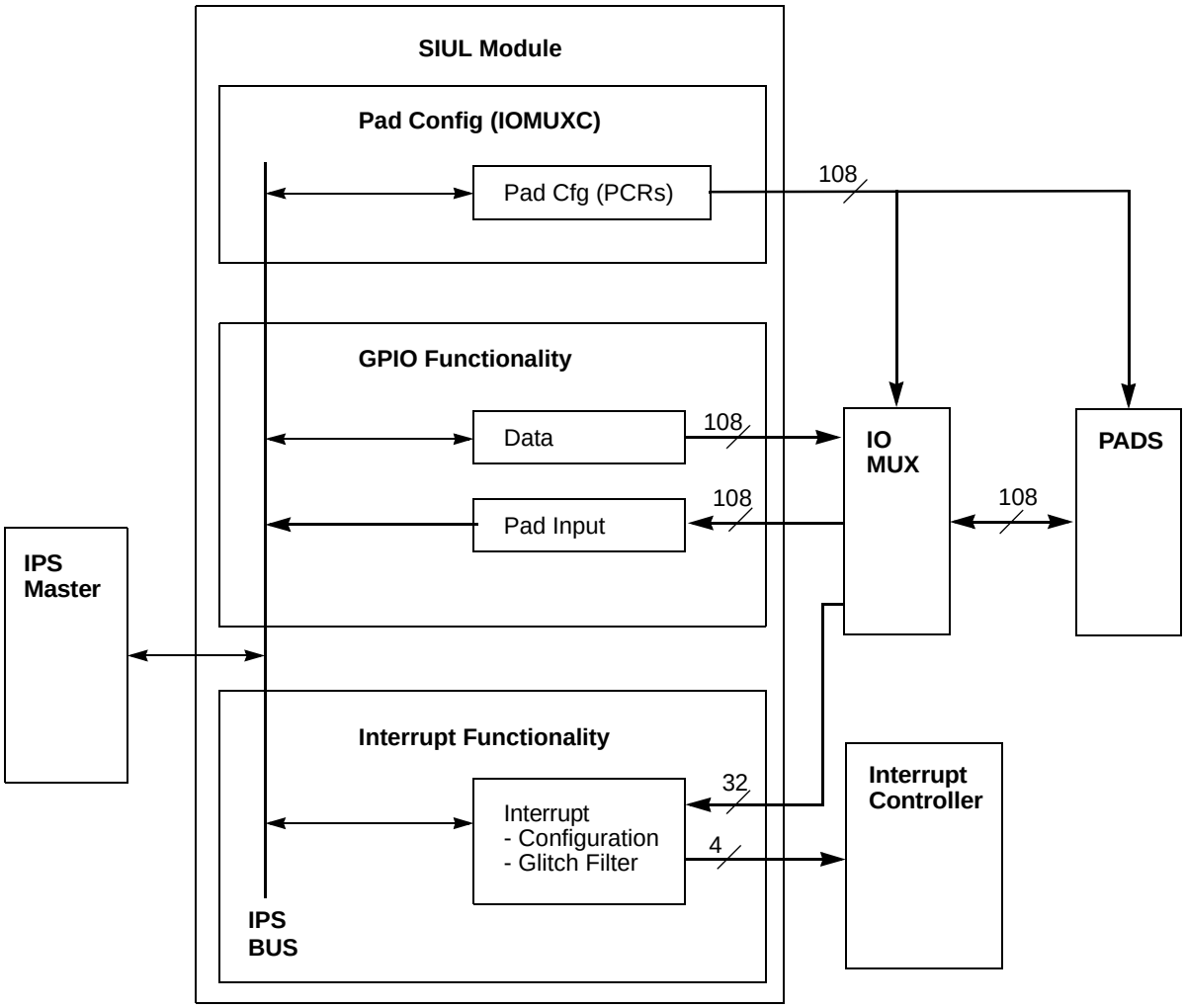


Figure 109. System Integration Unit Lite block diagram

11.3 Features

The System Integration Unit Lite provides these features:

- GPIO
  - GPIO function on up to 106 I/O pins
  - Dedicated input and output registers for each GPIO pin
- External interrupts
  - 4 system interrupt vectors for up to 32 interrupt sources
  - 32 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

11.3.1 Register protection

Most of the configuration registers of the System Integration Unit Lite are protected from accidental writes, see *Appendix A: Registers Under Protection*.

11.4 External signal description

The pad configuration allows flexible, centralized control of the pin electrical characteristics of the MCU with the GPIO control providing centralized general purpose I/O for an MCU that multiplexes GPIO with other signals at the I/O pads. These other signals, or alternate functions, will normally be the peripherals functions. The internal multiplexing allows user selection of the input to chip-level signal multiplexers. Each GPIO port communicates via 16 I/O channels. In order to use the pad as a GPIO, the corresponding Pad Configuration Registers (PCR[0:107]) for all pads used in the port must be configured as GPIO rather than as the alternate pad function.

Table 102 lists the external pins used by the SIUL.

Table 102. SIUL signal properties

GPIO category	Name	I/O direction	Function
System configuration	GPIO[0:19], GPIO[22], GPIO[35:62] GPIO[77:107]	Input/Output	General-purpose input/output
	GPIO[23:34], GPIO[63:76]	Input	Analog precise channel pins
External interrupt	EIRQ[0:31]	Input	Pins with External Interrupt Request functionality. Please refer to the signal description chapter of this reference manual for details.

11.4.1 Detailed signal descriptions

General-purpose I/O pins (GPIO[0:107])

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an input (GPDI*n_n*) or output (GPDO*n_n*) register.

See *Section : GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])* and *Section : GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])*.

External interrupt request input pins (EIRQ[0:31])

The EIRQ[0:31] are connected to the SIUL inputs. Rising or falling edge events are enabled by setting the corresponding bits in the “n” SIUL_IREER or the SIUL_IFEER register. See *Section : Interrupt Rising-Edge Event Enable Register (IREER)* and *Section : Interrupt Falling-Edge Event Enable Register (IFEER)*.

11.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

11.5.1 SIUL memory map

Table 103 lists the SIUL registers.

Table 103. SIUL memory map

Offset from SIUL_BASE (0xC3F9_0000)	Register	Location
0x0000–0x0003	Reserved	
0x0004	MCU ID Register #1 (MIDR1)	*on page 277*
0x0008	MCU ID Register #2 (MIDR2)	*on page 279*
0x000C–0x0013	Reserved	
0x0014	Interrupt Status Flag Register (ISR)	*on page 280*
0x0018	Interrupt Request Enable Register (IRER)	*on page 280*
0x001C–0x0027	Reserved	
0x0028	Interrupt Rising-Edge Event Enable Register (IREER)	*on page 281*
0x002C	Interrupt Falling-Edge Event Enable Register (IFEER)	*on page 281*
0x0030	Interrupt Filter Enable Register (IFER)	*on page 282*
0x0034–0x003F	Reserved	
0x0040–0x0118	Pad Configuration Registers (PCR[0:107])	*on page 282*
0x0118–0x04FF	Reserved	
0x0500–0x0524	Pad Selection for Multiplexed Inputs registers (PSMI[0_3:36_39])	*on page 284*
0x0528–0x05FF	Reserved	

Table 103. SIUL memory map (continued)

Offset from SIUL_BASE (0xC3F9_0000)	Register	Location
0x0600–0x0668	GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])	*on page 288*
0x066C–0x07FF	Reserved	
0x0800–0x0868	GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])	*on page 288*
0x086C–0x0BFF	Reserved	
0x0C00–0x0C0C	Parallel GPIO Pad Data Out register 0–3 (PGPDO[0:3])	*on page 289*
0x0C10–0x0C3F	Reserved	
0x0C40–0x0C4C	Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])	*on page 289*
0x0C50–0x0C7F	Reserved	
0x0C80–0x0C98	Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])	*on page 290*
0x0C9C–0x0FFF	Reserved	
0x1000–0x107C	Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31])	*on page 291*
0x1080	Interrupt Filter Clock Prescaler Register (IFCPR)	*on page 292*
0x1084–0x3FFF	Reserved	

Note: A transfer error will be issued when trying to access completely reserved register space.

11.5.2 Register description

This section describes in address order all the SIUL registers. Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of register is MSB = 0, however the numbering of internal field is LSB = 0, for example PARTNUM[5] = MIDR1[10].

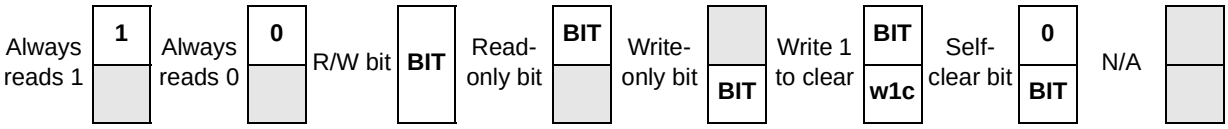


Figure 110. Key to register fields

MCU ID Register #1 (MIDR1)

This register contains the part number and the package ID of the device.

Figure 111. MCU ID Register #1 (MIDR1)

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM[15:0]															
W																
Reset	0	1	0	1	0	1	1	0	0	0	0	0	0	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSP	PKG[4:0]					0	0	MAJOR_MASK[3:0](1)				MINOR_MASK[3:0]1			
W																
Reset	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0

1. See *Table 104*.

Table 104. MIDR1 field descriptions

Field	Description
PARTNUM[15:0]	MCU Part Number Device part number of the MCU. 0101_0110_0000_0101: 768 KB, single core 0101_0110_0000_0110: 1 MB, single core 0101_0110_1010_0101: 768 KB, dual core 0101_0110_1010_0110: 1 MB, dual core For the full part number this field needs to be combined with MIDR2.PARTNUM[23:16]
CSP	Always reads back 0
PKG[4:0]	Package Settings Can by read by software to determine the package type that is used for the particular device: 01001: 100-pin LQFP 01101: 144-pin LQFP
MAJOR_MASK[3:0]	Major Mask Revision Counter starting at 0x0. Incremented each time a resynthesis is done.
MINOR_MASK[3:0]	Minor Mask Revision Counter starting at 0x0. Incremented each time a mask change is done.

MCU ID Register #2 (MIDR2)

This register contains additional configuration information about the device.

Figure 112. MCU ID Register #2 (MIDR2)

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SF	FLASH_SIZE_1[3:0]				FLASH_SIZE_2[3:0]			0	0	0	0	0	0	0	0
W																
Reset	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PARTNUM[23:16]								0	0	0	EE	0	0	0	0
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 105. MIDR2 field descriptions

Field	Description
SF	Manufacturer 0: Reserved 1: ST
FLASH_SIZE_1[3:0]	Coarse granularity for Flash memory size Needs to be combined with FLASH_SIZE_2 to calculate the actual memory size. 0101: 512 KB 0110: 1 MB Other values are reserved.
FLASH_SIZE_2[3:0]	Fine granularity for Flash memory size Needs to be combined with FLASH_SIZE_1 to calculate the actual memory size. 0000: 0 x (FLASH_SIZE_1 / 8) 0010: 2 x (FLASH_SIZE_1 / 8) 0100: 4 x (FLASH_SIZE_1 / 8) Other values are reserved.
PARTNUM[23:16]	ASCII character in MCU Part Number 0x50: P family (Steering)
EE	Data Flash present 0: No Data Flash present 1: Data Flash present

Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

Figure 113. Interrupt Status Flag Register (ISR)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EIF[31:16]															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF[15:0]															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 106. ISR field descriptions

Field	Description
EIF*n*	External Interrupt Status Flag *n* This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER*n*), EIF*n* causes an interrupt request. 0: No interrupt event has occurred on the pad. 1: An interrupt event as defined by IREER*n* and IFEER*n* has occurred.

Interrupt Request Enable Register (IRER)

This register enables the interrupt messaging to the interrupt controller.

Figure 114. Interrupt Request Enable Register (IRER)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IRE[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IRE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 107. IRER field descriptions

Field	Description
IRE*n*	External Interrupt Request Enable *n* 0: Interrupt requests from the corresponding EIF*n* bit are disabled. 1: A set EIF*n* bit causes an interrupt request.

Interrupt Rising-Edge Event Enable Register (IREER)

This register allows rising-edge triggered events to be enabled on the corresponding interrupt pads.

Figure 115. Interrupt Rising-Edge Event Enable Register (IREER)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IREE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IREE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 108. IREER field descriptions

Field	Description
IREE*n*	Enable rising-edge events to cause the EIF*n* bit to be set. 0: Rising-edge event disabled 1: Rising-edge event enabled

Interrupt Falling-Edge Event Enable Register (IFEER)

This register allows falling-edge triggered events to be enabled on the corresponding interrupt pads.

Figure 116. Interrupt Falling-Edge Event Enable Register (IFEER)

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IFEE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IFEE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 109. IFEER field descriptions

Field	Description
IFEE*n*	Enable falling-edge events to cause the EIF*n* bit to be set. 0: Falling-edge event disabled 1: Falling-edge event enabled

Note: If both the IREER.IREE and IFEER.IFEE bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt will never be set.

Interrupt Filter Enable Register (IFER)

This register enables a digital filter counter on the corresponding interrupt pads to filter out glitches on the inputs.

Figure 117. Interrupt Filter Enable Register (IFER)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IFE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IFE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 110. IFER field descriptions

Field	Description
IFE*n*	Enable digital glitch filter on the interrupt pad input. 0: Filter disabled 1:Filter enabled

Pad Configuration Registers (PCR[0:107])

The Pad Configuration Registers allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR controls the characteristics of a single pad.

Figure 118. Pad Configuration Registers 0–108 (PCR[0:108])

Base + 0x0040 (PCR0)

Address: ... Access: User read/write

Base + 0x0118 (PCR108) 109 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC	0	PA[1:0]		OBE	IBE	0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. See *Table 112*.

Note: 16/32-bit access is supported for the PCR[0:108] registers.

Table 111. PCR[0:107] field descriptions

Field	Description
SMC	Safe Mode Control This bit supports the overriding of the automatic deactivation of the output buffer of the associated pad upon entering Safe mode of the device. 0: In Safe mode, output buffer of the pad disabled 1: In Safe mode, output buffer remains functional
APC	Analog Pad Control This bit enables the usage of the pad as analog input. 0: Analog input path from the pad is gated and cannot be used. 1: Analog input path switch can be enabled by the ADC.
PA[1:0]	Pad Output Assignment This field selects the function that is allowed to drive the output of a multiplexed pad.The PA field size can vary from 0 to 2 bits, depending on the number of output functions associated with this pad. 00: Alternative mode 0: GPIO 01: Alternative mode 1 (see *Chapter 3: Signal Description*) 10: Alternative mode 2 (see *Chapter 3: Signal Description*) 11: Alternative mode 3 (see *Chapter 3: Signal Description*) The number of bits in the PA bitfield depends of the number of actual alternate functions provided for each pad. Please see the SPC56xP60/54 *Datasheet (*SPC56xP60/54*)*.
OBE	Output Buffer Enable This bit enables the output buffer of the pad in case the pad is in GPIO mode. 0: Output buffer of the pad disabled when PA = 00 1: Output buffer of the pad enabled when PA = 00
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 0: Input buffer of the pad disabled 1: Input buffer of the pad enabled
ODE	Open Drain Output Enable This bit controls output driver configuration for the pads connected to this signal. Either open drain or push/pull driver configurations can be selected. This feature applies to output pads only. 0: Open drain enable signal negated for the pad 1: Open drain enable signal asserted for the pad
SRC	Slew Rate Control 0: Slowest configuration 1: Fastest configuration
WPE	Weak Pull Up/Down Enable This bit controls whether the weak pull up/down devices are enabled/disabled for the pad connected to this signal. 0: Weak pull device enable signal negated for the pad 1: Weak pull device enable signal asserted for the pad
WPS	Weak Pull Up/Down Select This bit controls whether weak pull up or weak pull down devices are used for the pads connected to this signal when weak pull up/down devices are enabled. 0: Pull down enabled 1: Pull up enabled

Table 112. PCR[*n*] reset value exceptions

Field	Description
PCR[2] PCR[3] PCR[4]	These registers correspond to the ABS[0], ABS[1], and FAB boot pins, respectively. Their default state is input, pull enabled. Their reset value is 0x0102.
PCR[20]	This register corresponds to the TDO pin. Its default state is ALT1, slew rate = 1. Its reset value is 0x0604.
PCR[21]	This register corresponds to the TDI pin. Its default state is input, pull enabled, pull selected, slew enabled. So its reset value is 0x0107.
PCR[108]	This register corresponds to the $\overline{\text{RESET}}$ pin. Its reset value is 0x0102.
PCR[*n*]	For other PCR[*n*] registers, the reset value is 0x0000.

In addition to the bit map above, the following *Table 113 [PCR bit implementation by pad type]* describes the PCR depending on the pad type (refer to *Section 3.4.3: Pin muxing* for pad types description). The bits in shaded fields are not implemented for the particular I/O type. The PA field selecting the number of alternate functions may or may not be present depending on the number of alternate functions actually mapped on the pad.

Table 113. PCR bit implementation by pad type

Pad type	PCR bit No.															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S, M, F (Pad with GPIO and digital alternate functionality) (1)		SMC	APC		PA [1:0]		OBE	IBE			ODE			SRC	WPE	WPS
I (Pad with GPIO and analog functionality)		SMC	APC		PA [1:0]		OBE	IBE			ODE			SRC	WPE	WPS

1. Only OBE, IBE, ODE, SRC, WPE, and WPS fields are implemented in PCR[108].

Pad Selection for Multiplexed Inputs registers (PSMI[0_3:36_39])

The purpose of the PSMI[0_3:36_39] registers is to allow connecting a single input pad to one of several peripheral inputs. Thus, it is possible to define different pads to be possible inputs for a certain peripheral function.

Figure 119. Pad Selection for Multiplexed Inputs registers (PSMI[0_3:36_39])

Address: Base + 0x0500 (PSMI0_3), Base + 0x0504 (PSMI4_7), Base + 0x0508 (PSMI8_11), Base + 0x050C (PSMI12_15), Base + 0x0510 (PSMI16_19), Base + 0x0514 (PSMI20_23), Base + 0x0518 (PSMI024_27), Base + 0x051C (PSMI28_31), Base + 0x0520 (PSMI32_35), Base + 0x0524 (PSMI36_39)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PADSEL0[3:0]				0	0	0	0	PADSEL1[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PADSEL2[3:0]				0	0	0	0	PADSEL3[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 114. PSMI[0_3:36_39] field descriptions

Field	Description
PADSEL0–3 ... PADSEL32–35	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function. See *Table 115.*.

Table 115. Pad selection

Register	PADSEL	Module	Port	PADSEL field(1)	Port name	LQFP pin		
						176-pin	144-pin	100-pin
PSMI0_3	PADSEL0	ctu0	EXT_IN	0b	C[13]	125	101	71
				1b	C[15]	148	124	85
	PADSEL1	dspi2	SCK	0b	A[0]	89	73	51
				1b	A[11]	144	120	82
	PADSEL2	dspi2	SIN	0b	A[2]	102	84	57
				1b	A[13]	160	136	95
	PADSEL3	dspi2	CS0	0b	A[3]	116	92	64
				1b	A[10]	142	118	81

Table 115. Pad selection (continued)

Register	PADSEL	Module	Port	PADSEL field[1]	Port name	LQFP pin		
						176-pin	144-pin	100-pin
PSMI4_7	PADSEL0	dspi3	SCK	00b	D[6]	42	34	23
				01b	D[11]	94	78	54
				10b	E[13]	141	117	—
	PADSEL1	dspi3	SIN	00b	D[7]	45	37	26
				01b	D[14]	129	105	73
				10b	E[15]	145	121	—
	PADSEL2	dspi3	CS0	00b	C[11]	96	80	55
				01b	D[10]	92	76	53
				10b	F[3]	176	139	—
	PADSEL3	eTimer0	ETC[4]	00b	A[4]	132	108	75
				01b	C[11]	96	80	55
				10b	B[14]	76	54	44
PSMI8_11	PADSEL0	eTimer0	ETC[5]	0b	C[12]	100	82	56
				1b	B[8]	55	47	31
	PADSEL1	eTimer1	ETC[0]	0b	A[4]	132	108	75
				1b	C[15]	148	124	85
	PADSEL2	eTimer1	ETC[1]	0b	C[13]	125	101	71
				1b	D[0]	149	125	86
	PADSEL3	eTimer1	ETC[2]	00b	B[0]	133	109	76
				01b	C[14]	127	103	72
				10b	D[1]	3	3	3
PSMI12_15	PADSEL0	eTimer1	ETC[3]	0b	B[1]	134	110	77
				1b	D[2]	168	140	97
				10b	F[12]	130	106	—
	PADSEL1	eTimer1	ETC[4]	0b	A[14]	175	143	99
				1b	C[3]	24	16	10
				10b	D[3]	152	128	89
				11b	F[13]	136	112	—
	PADSEL2	eTimer1	ETC[5]	0b	A[5]	22	14	8
				1b	A[15]	176	144	100
				10b	D[4]	153	129	90
	PADSEL3							
PSMI16-19	UNIMPLEMENTED							

Table 115. Pad selection (continued)

Register	PADSEL	Module	Port	PADSEL field[1]	Port name	LQFP pin		
						176-pin	144-pin	100-pin
PSMI20-23	UNIMPLEMENTED							
PSMI24-27	UNIMPLEMENTED							
PSMI28-31	PADSEL0	UNIMPLEMENTED						
	PADSEL1	UNIMPLEMENTED						
	PADSEL2	UNIMPLEMENTED						
	PADSEL3	Lin0	RX	0b	B[3]	140	116	80
				1b	B[7]	51	43	29
PSMI32-35	PADSEL0	Lin1	RX	0b	B[13]	68	60	42
				1b	D[12]	123	99	70
				10b	F[15]	137	113	—
	PADSEL1	Can1	RX	0b	C[13]	125	110	71
				1b	G[9]	95	79	—
	PADSEL2	Dspi4	SCK	0b	B[3]	140	116	80
				1b	C[5]	21	13	7
				10b	G[4]	124	100	—
	PADSEL3	Dspi4	SIN	0b	A[9]	158	134	94
				1b	C[7]	23	15	9
				10b	G[2]	126	102	—
PSMI36-39	PADSEL0	Dspi4	CS0	0b	A[8]	20	12	6
				1b	C[9]	147	123	84
				10b	G[5]	103	85	—
	PADSEL1	UNIMPLEMENTED						
	PADSEL2	UNIMPLEMENTED						
	PADSEL3	UNIMPLEMENTED						

1. Values not listed are reserved.

GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])

These registers can be used to set or clear a single GPIO pad with a byte access.

Figure 120. Port GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])

Base + 0x0600 (GPDO0_3)
Address: ... Access: User read/write
Base + 0x0668 (GPDO104_107) 27 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO[0]	0	0	0	0	0	0	0	PDO[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO[2]	0	0	0	0	0	0	0	PDO[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 116. GPDO[0_3:104_107] field descriptions

Field	Description
PDO[*n*]	Pad Data Out This bit stores the data to be driven out on the external GPIO pad controlled by this register. 0: Logic low value is driven on the corresponding GPIO pad when the pad is configured as an output. 1: Logic high value is driven on the corresponding GPIO pad when the pad is configured as an output.

GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])

These registers can be used to read the GPIO pad data with a byte access.

Figure 121. GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])

Base + 0x0800 (GPDI0_3)
Address: ... Access: User read-only
Base + 0x0868 (GPDI104_107) 27 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI[0]	0	0	0	0	0	0	0	PDI[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI[2]	0	0	0	0	0	0	0	PDI[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 117. GPDI[0_3:104_107] field descriptions

Field	Description
PDI[*x*]	Pad Data In This bit stores the value of the external GPIO pad associated with this register. 0: The value of the data in signal for the corresponding GPIO pad is logic low. 1: The value of the data in signal for the corresponding GPIO pad is logic high.

Parallel GPIO Pad Data Out register 0–3 (PGPDO[0:3])

These registers set or clear the respective pads of the device.

Figure 122. Parallel GPIO Pad Data Out register 0–3(PGPDO[0:3])

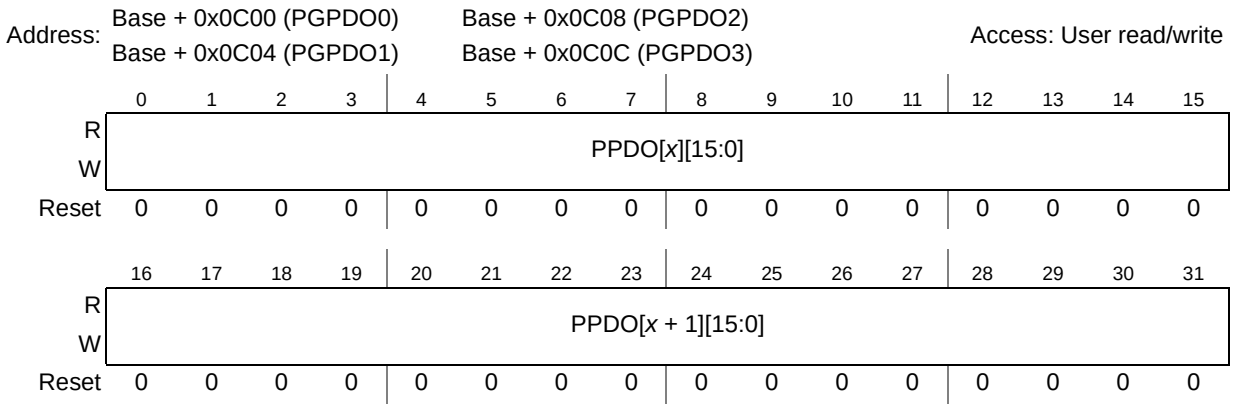


Table 118. PGPDO0_3 field descriptions

Field	Description
PPDO[*x*]	Parallel Pad Data Out Write or read the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107]). The *x* and bit index define which PPDO register bit is equivalent to which PDO register bit according to the following equation: PPDO[*x*][*y*] = PDO[(*x* * 16) + *y*]

Note: The PGPDO registers access the same physical resource as the PDO and MPGPDO address locations. Some examples of the mapping:

PPDO[0][0] = PDO[0]

PPDO[2][0] = PDO[32]

Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])

These registers hold the synchronized input value from the pads.

Figure 123. Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])

Address: Base + 0x0C40 (PGPDI0) Base + 0x0C45 (PGPDI2)
Base + 0x0C44 (PGPDI1) Base + 0x0C4C (PGPDI3)
Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PPDI[*x*][15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PPDI[*x* + 1][15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 119. PGPDI[0:3] field descriptions

Field	Description
PPDI[*x*]	Parallel Pad Data In Read the current pad value. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107]). The *x* and bit index define which PPDI register bit is equivalent to which PDI register bit according to the following equation: PPDI[*x*][*y*] = PDI[(*x* * 16) + *y*]

Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])

This register can be used to selectively modify the pad values associated to PPDO[*x*][15:0]. The MPGPDO[*x*] register may only be accessed with 32-bit writes. 8-bit or 16-bit writes will not modify any bits in the register and cause a transfer error response by the module. Read accesses will return 0.

Figure 124. Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])

Address: Base + 0x0C80 (MPGPDO0) Base + 0x0C80 (MPGPDO4)
Base + 0x0C84 (MPGPDO1) Base + 0x0C84 (MPGPDO5)
Base + 0x0C88 (MPGPDO2) Base + 0x0C98 (MPGPDO6)
Base + 0x0C8C (MPGPDO3)
Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	MASK[*x*][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	MPPDO[*x*][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 120. MPGPDO[0:6] field descriptions

Field	Description
MASK[*x*] [15:0]	Mask Field Each bit corresponds to one data bit in the MPPDO[x] field at the same bit location. 0: The associated bit value in the MPPDO[*x*] field is ignored. 1: The associated bit value in the MPPDO[*x*] field is written.
MPPDO[*x*] [15:0]	Masked Parallel Pad Data Out Write the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107]). The x and bit index define which MPPDO register bit is equivalent to which PDO register bit according to the following equation: MPPDO[*x*][*y*] = PDO[(*x* * 16) + *y*]

Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31])

These registers configure the filter counter associated with each digital glitch filter.

Figure 125. Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31])

Base + 0x1000 (IFMC0)

Address: ... Access: User read/write

Base + 0x107C (IFMC31) 32 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MAXCNT*x*[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 121. IFMC[0:31] field descriptions

Field	Description
MAXCNT*x* [3:0]	Maximum Interrupt Filter Counter setting. Filter Period = (T_{CK} × MAXCNT*x*) + (*n* × T_{CK}) Where *n* can be –2 to 3 MAXCNT*x* can be 0 to 15 T_{CK}: Prescaled Filter Clock Period, which is IRC clock prescaled to IFCP value T_{IRC}: Basic Filter Clock Period: 62.5 ns (f_{IRC} = 16 MHz)

Interrupt Filter Clock Prescaler Register (IFCPR)

This register configures a clock prescaler that selects the clock for all digital filter counters in the SIUL.

Figure 126. Interrupt Filter Clock Prescaler Register (IFCPR)

Address: Base + 0x1080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFCP[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 122. IFCPR field descriptions

Field	Description
IFCP [3:0]	Interrupt Filter Clock Prescaler setting Prescaled Filter Clock Period = T_{IRC} × (IFCP + 1) T_{IRC} is the internal oscillator period. IFCP can be 0 to 15.

11.6 Functional description

11.6.1 General

This section provides a functional description of the System Integration Unit Lite.

11.6.2 Pad control

The SIUL controls the configuration and electrical characteristic of the device pads. It provides a consistent interface for all pads, both on a by-port and a by-bit basis. The SIUL allows each pad to be configured as either a General Purpose Input Output pad (GPIO), and as one or more alternate functions (input or output). The pad configuration registers (PCR*n*, see *Section : Pad Configuration Registers (PCR[0:107])*) allow software control of the static electrical characteristics of external pins with a single write. These configure the following pad features:

- Open drain output enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode behavior configuration

11.6.3 General purpose input and output pads (GPIO)

The SIUL allows each pad to be configured as either a General Purpose Input Output pad (GPIO), and as one or more alternate functions (input or output), the function of which is normally determined by the peripheral that uses the pad.

The SIUL manages up to 106 GPIO pads organized as ports that can be accessed for data reads and writes as 32-bit, 16-bit or 8-bit.

As shown in *Figure 127*, all port accesses are identical with each read or write being performed only at a different location to access a different port width.

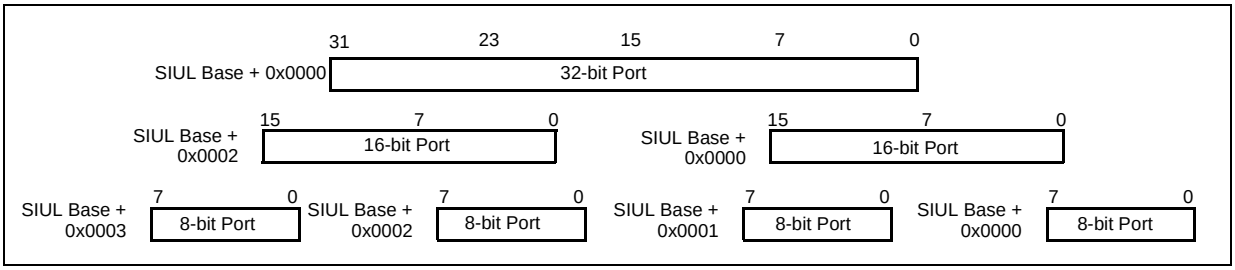


Figure 127. Data port example arrangement showing configuration for different port width accesses

This implementation requires that the registers are arranged in such a way as to support this range of port widths without having to split reads or writes into multiple accesses.

The SIUL has separate data input (GPDI*n*_*n*, see *Section : GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])*) and data output (GPDO*n*_*n*, see *Section : GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])*) registers for all pads, allowing

the possibility of reading back an input or output value of a pad directly. This supports the ability to validate what is present on the pad rather than merely confirming the value that was written to the data register by accessing the data input registers.

The data output registers support both read and write operations to be performed.

The data input registers support read access only.

When the pad is configured to use one of its alternate functions, the data input value reflect the respective value of the pad. If a write operation is performed to the data output register for a pad configured as an alternate function (non GPIO), this write will not be reflected by the pad value until reconfigured to GPIO.

The allocation of what input function is connected to the pin is defined by the PSMI registers (see *Section : Pad Selection for Multiplexed Inputs registers (PSMI[0_3:36_39])*).

11.6.4 External interrupts

The SIUL supports 32 external interrupts, EIRQ[0:31]. The signal description chapter of this reference manual provides a map of the external interrupts.

The SIUL supports four interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

Refer to *Figure 128* for an overview of the external interrupt implementation.

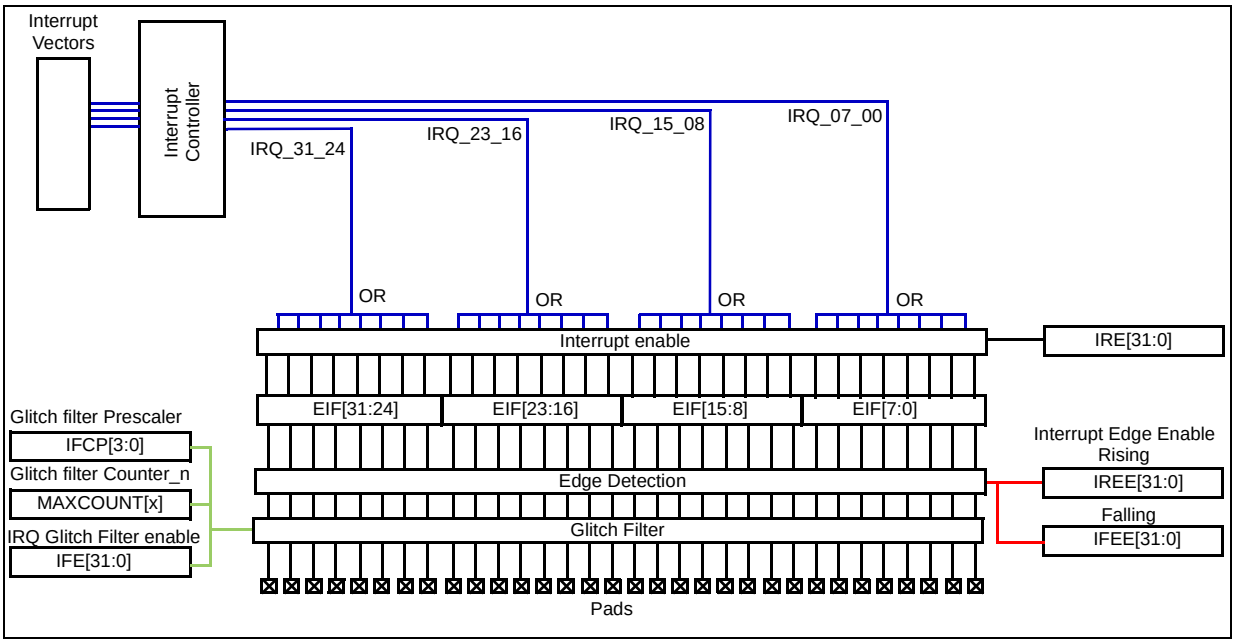


Figure 128. External interrupt pad diagram

External interrupt management

Each interrupt can be enabled or disabled independently. This can be performed using the IRER (see *Section : Interrupt Request Enable Register (IRER)*). A pad defined as an external interrupt can be configured to recognize interrupts with an active rising edge, an

active falling edge or both edges being active. A setting of having both edge events disabled is reserved and should not be configured.

The active EIRQ edge is controlled through the configuration of the registers IREER and IFEER.

Each external interrupt supports an individual flag that is held in the ISR (see *Section : Interrupt Status Flag Register (ISR)*). This register is a write-1-to-clear register type, preventing inadvertent overwriting of other flags in the same register.

11.7 Pin muxing

For pin muxing, please refer to *Chapter 3: Signal Description* of this reference manual.

12 e200z0 and e200z0h Core

12.1 Overview

The SPC56xP60/54 microcontroller implements the e200z0h core.

The e200 processor family is a set of CPU cores built on the Power Architecture technology. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z0 and e200z0h processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in some cases.

The e200z0 core is a single-issue, 32-bit Power Architecture technology VLE-only design with 32-bit general purpose registers (GPRs). Implementing only the VLE (variable-length encoding) APU provides improved code density. All arithmetic instructions that execute in the core operate on data in the GPRs.

12.2 Features

The following is a list of some of the key features of the e200z0 and e200z0h cores:

- 32-bit Power Architecture technology VLE-only programmer's model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Target Buffer (e200z0h only)
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data bus interface units (BIUs) (e200z0h only)
- Supports instruction and data access via a unified 32-bit Instruction/Data BIU (e200z0 only)
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big-endian support only
  - Misaligned access support
  - Zero load-to-use pipeline bubbles for aligned transfers
- Power management
  - Low power design
  - Dynamic power management of execution units

- Testability
  - Synthesizeable, full MuxD scan design
  - ABIST/MBIST for optional memory arrays

12.2.1 Microarchitecture summary

The e200z0 processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), an 8 × 32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching from the BTB is performed to accelerate certain taken branches in the e200z0. Prefetched instructions are placed into an instruction buffer with 4 entries (2 entries in e200z0), each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches that are not taken execute in a single clock. Branches with successful target prefetching have an effective execution time of one clock on e200z0h. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

Block diagram

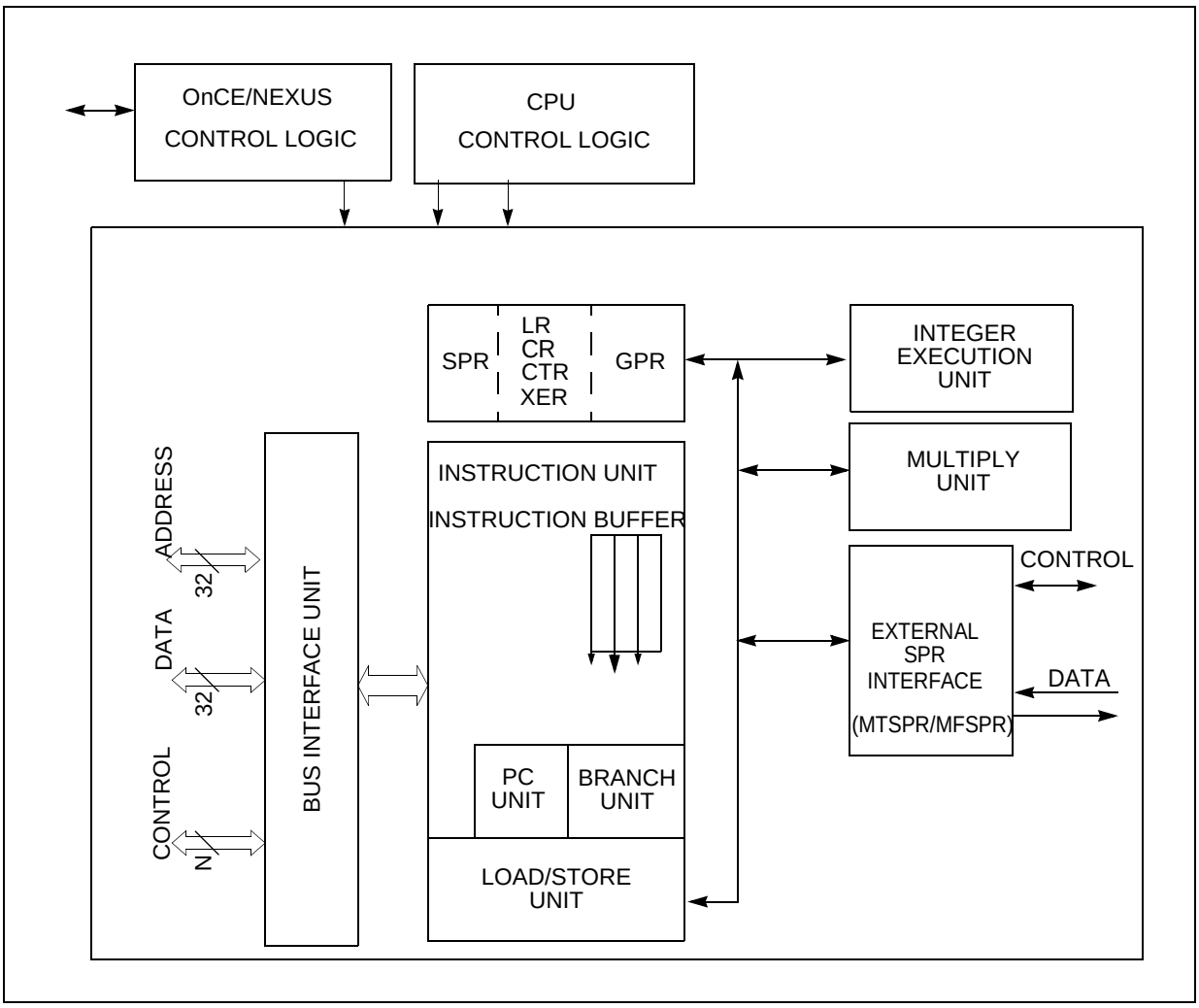


Figure 129. e200z0 block diagram

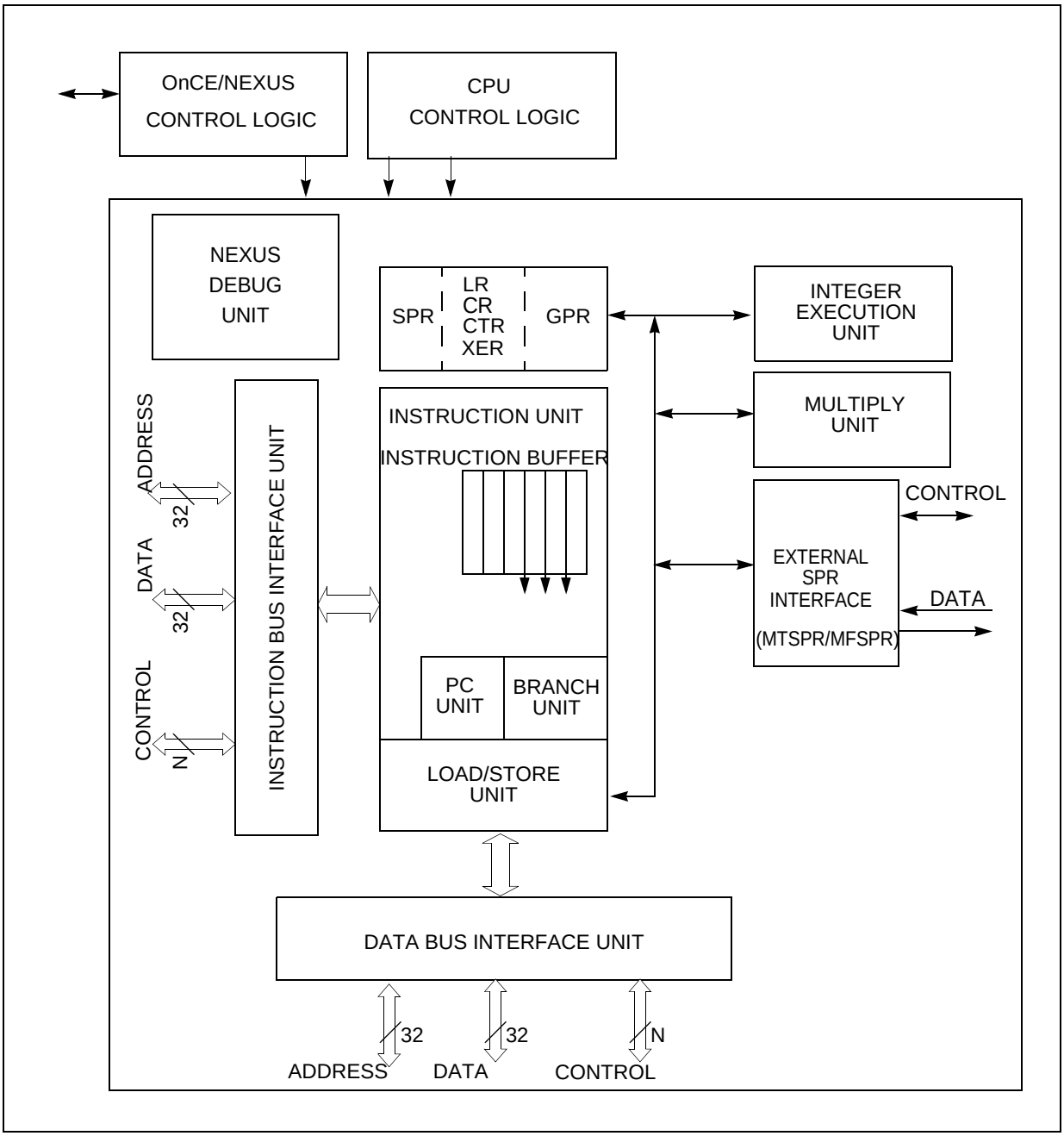


Figure 130. e200z0h block diagram

Instruction unit features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or as many as two 16-bit VLE instructions per clock
- Instruction buffer with 4 entries in e200z0h, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Instruction buffer with 2 entries in e200z0, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

Integer unit features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- 8 × 32 hardware multiplier array supports 1 to 4 cycle 32 × 32 → 32 multiply (early out)

Load/Store unit features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory (dedicated memory interface on e200z0h)

e200z0h system bus features

The features of the e200z0h System Bus interface are as follows:

- Independent Instruction and Data Buses
- AMBA AHB Lite Rev 2.0 Specification with support for ARM v6 AMBA Extensions
  - Exclusive Access Monitor
  - Byte Lane Strobes
  - Cache Allocate Support
- 32-bit address bus plus attributes and control on each bus
- 32-bit read data bus for Instruction Interface
- Separate uni-directional 32-bit read data bus and 32-bit write data bus for Data Interface
- Overlapped, in-order accesses

Nexus 2+ features

The Nexus 2+ module is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and Class 4 features available. The following features are implemented:

- Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.
- Ownership Trace via Ownership Trace Messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to the processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint Messaging via the auxiliary interface
- Watchpoint Trigger enable of Program Trace Messaging
- Auxiliary interface for higher data input/output
  - Configurable (min/max) Message Data Out pins (**nex_mdo[n:0]**)
  - One (1) or two (2) Message Start/End Out pins (**nex_mseo_b[1:0]**)
  - One (1) Read/Write Ready pin (**nex_rdy_b**) pin
  - One (1) Watchpoint Event pin (**nex_evto_b**)
  - One (1) Event In pin (**nex_evti_b**)
  - One (1) MCKO (Message Clock Out) pin
- Registers for Program Trace, Ownership Trace and Watchpoint Trigger control
- All features controllable and configurable via the JTAG port

12.3 Core registers and programmer's model

This section describes the registers implemented in the e200z0 and e200z0h cores. It includes an overview of registers defined by the Power Architecture technology, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Specification.

The Power Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 131 and *Figure 133* show the e200 register set, including the registers that are accessible while in supervisor mode and the registers that are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

Note: e200z0 and e200z0h is a 32-bit implementation of the Power Architecture specification.

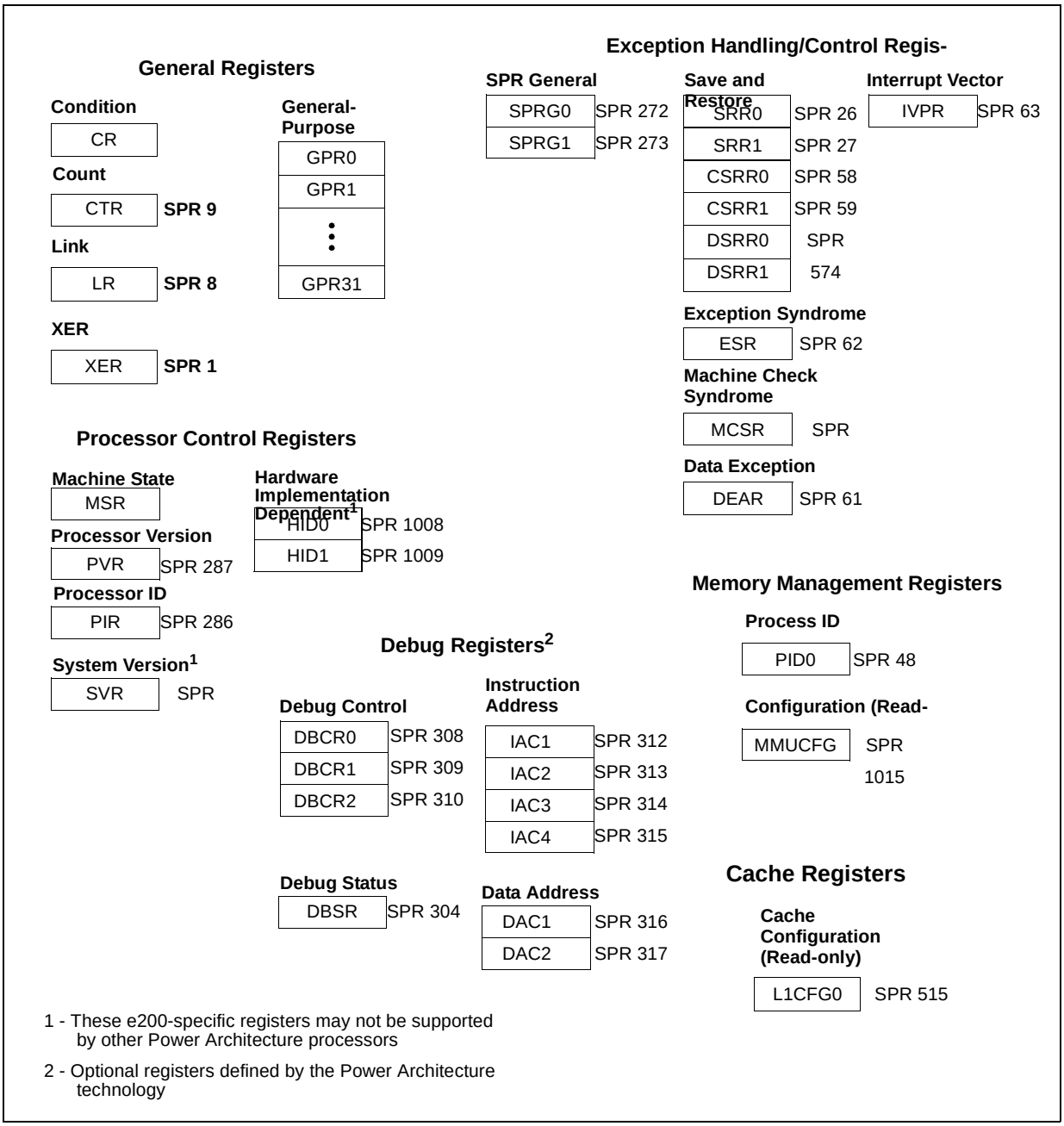


Figure 131. e200z0 Supervisor mode programmer's model

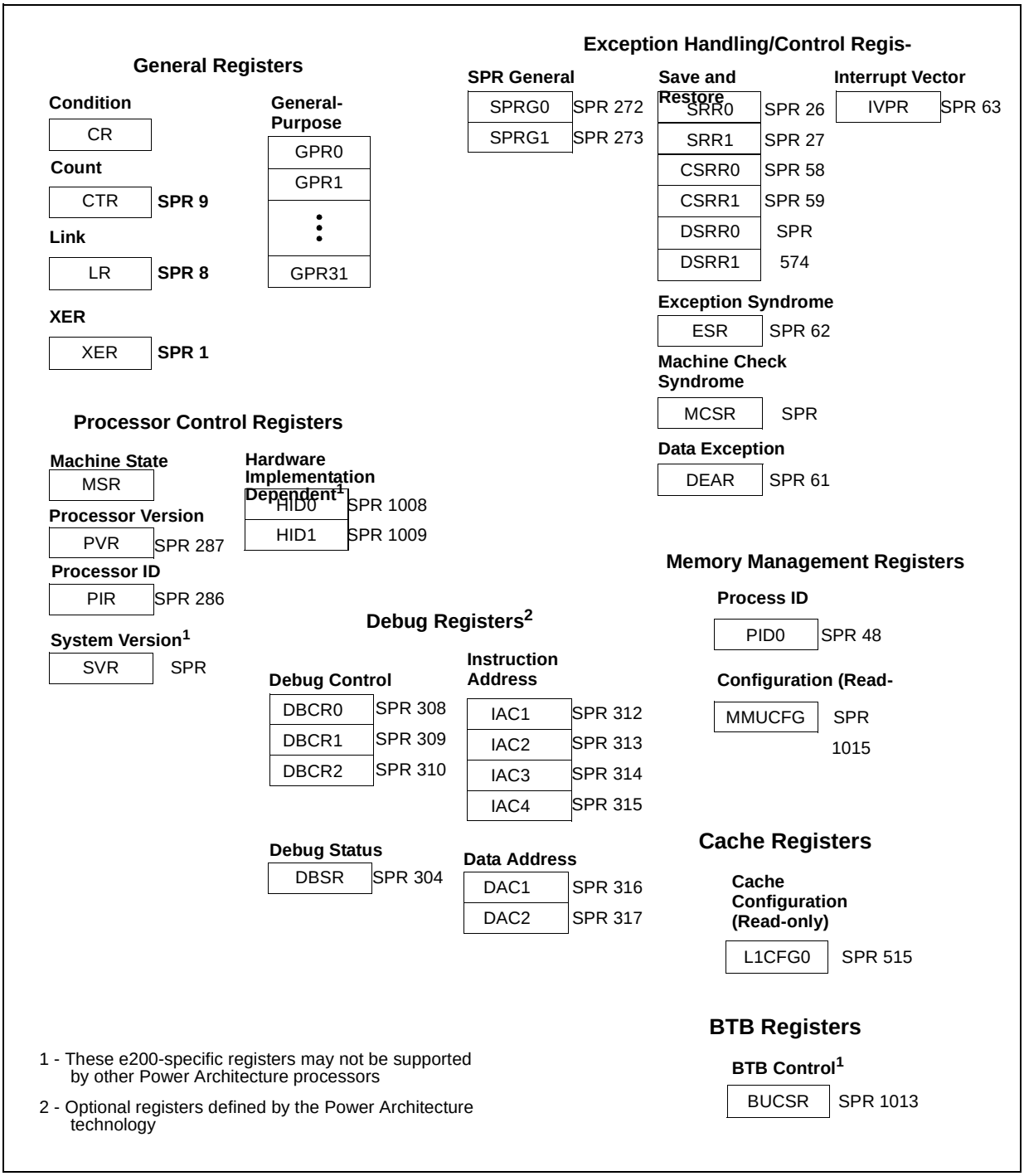


Figure 132. e200z0h Supervisor mode programmer's model

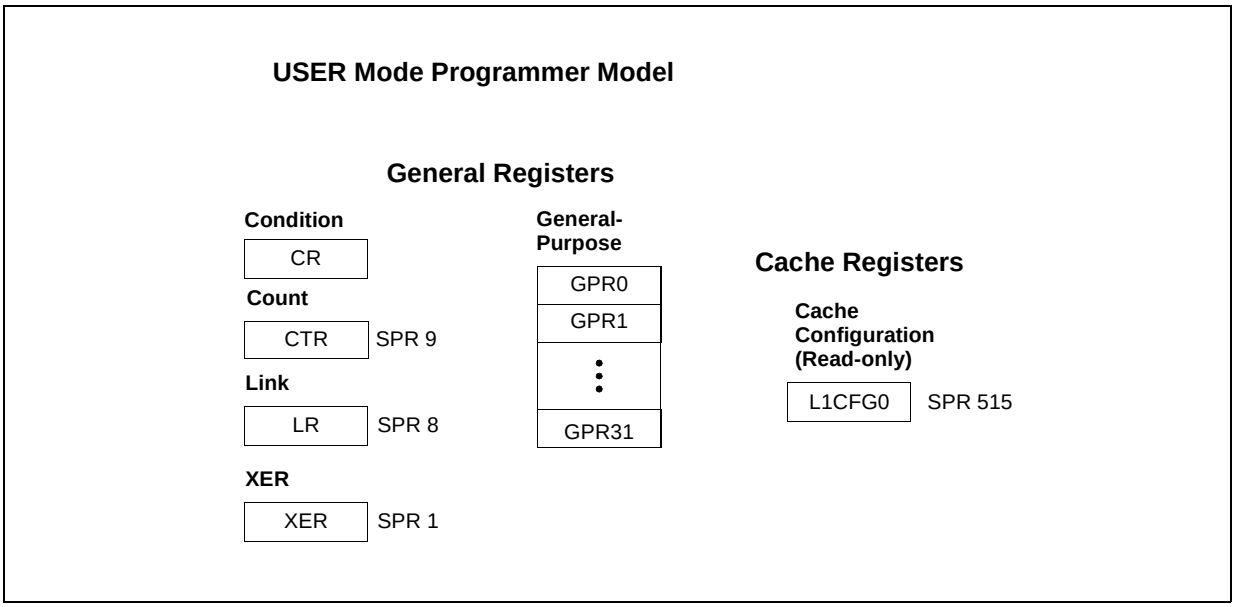


Figure 133. e200 User mode program model

12.3.1 Unimplemented SPRs and read-only SPRs

e200 fully decodes the SPR field of the **mfspr** and **mtspr** instructions. If the SPR specified is undefined and not privileged, an illegal instruction exception is generated. If the SPR specified is undefined and privileged and the CPU is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is undefined and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

For the **mtspr** instruction, if the SPR specified is read-only and not privileged, an illegal instruction exception is generated. If the SPR specified is read-only and privileged and the core is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is read-only and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

12.4 Instruction summary

The e200z0 core supports Power Architecture technology VLE instructions..

13 Peripheral Bridge (PBRIDGE)

13.1 Introduction

The peripheral bridge (PBRIDGE) is the interface between the system bus and on-chip peripherals. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

The PBRIDGE will automatically manage peripheral access requests from software. In other words, you can access a peripheral using its memory-mapped interface without having to configure the PBRIDGE. However, the PBRIDGE also supports special peripheral access, such as memory protection, that is available for you to use if you need it.

This device includes two instantiations of the PBRIDGE. These are called PBRIDGE_0 and PBRIDGE_1.

13.2 Block interface

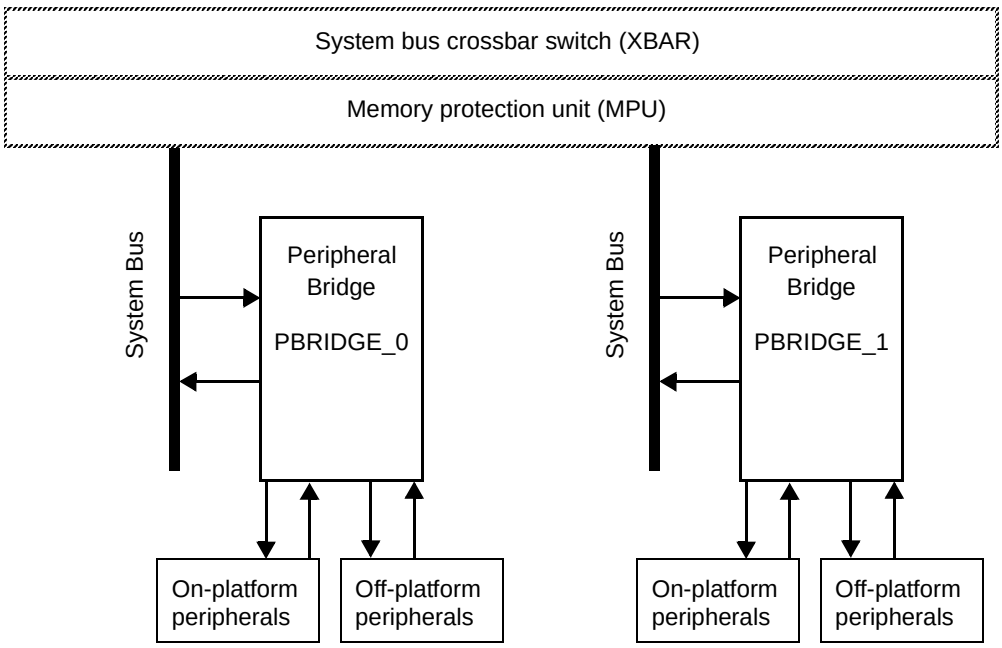


Figure 134. PBRIDGE interface

13.3 Features

The following list summarizes the key features of the PBRIDGE:

- Includes a 32-bit address bus and 32-bit data bus
- Supports 32-bit slave peripherals (byte, halfword, and word reads and writes are supported by the bridge)
- Provides configurable per-master access protections
- Provides configurable per-peripheral access protections for both on-platform and off-platform peripherals

13.4 Memory map and register description

13.4.1 Register access

All registers are 32-bit registers and can only be accessed in supervisor mode by trusted bus masters. Additionally, these registers must only be read from or written to by a 32-bit aligned access.

Two system clock cycles are required for read accesses and three system clock cycles are required for write accesses to the PBRIDGE registers.

13.4.2 Memory map

Each register in the PBRIDGE module has a size of 32 bits. The registers are listed in *Table 123* . The memory map organization is shown in *Table 124*. The organizational hierarchy is as follows:

- The module has multiple registers with the same register name (MPROT, PACR, OPACR), each at a different address offset.
- Each register has multiple similarly-named fields, each with a different number.
- Each field has subfields as defined elsewhere in this section.

Accesses to registers or register fields marked as reserved will return undefined data on reads, and will be ignored on writes.

Table 123. PBRIDGE registers

Offset from PBRIDGE_BASE PBRIDGE_0 = 0xFFF0_0000 PBRIDGE_1 = 0x8FF0_0000	Register	Location
0x0000–0x0007 (1)	Master Protection Registers (MPROT)	*on page 307*
0x0008–0x001F	Reserved	
0x0020–0x003F (1)	Peripheral Access Control Registers (PACR)	*on page 309*
0x0040–0x006F (1)	Off-Platform Peripheral Access Control Registers (OPACR)	*on page 310*
0x0070–0x3FFF	Reserved	

1. This memory range contains reserved areas. See *Table 124*.

Table 124. PBRIDGE memory map

Address offset	Register name	Bit numbers							
		0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
0x0000	MPROT	MPROT0	MPROT1	MPROT2	MPROT3	Reserved	MPROT5	MPROT6	Reserved
0x0004		MPROT8	MPROT9	Reserved					
0x0020	PACR	PACR0	PACR1(1)	Reserved		PACR4	Reserved		
0x0024		Reserved	PACR9	Reserved				PACR14	PACR15
0x0028		PACR16	PACR17(1)	PACR18	Reserved				
0x002C		Reserved							
0x0030	Reserved								
0x0034	Reserved								
0x0038	Reserved								
0x003C	Reserved								
0x0040	OPACR	Reserved				OPACR4	OPACR5	OPACR6	OPACR7
0x0044		OPACR8(2)	Reserved						
0x0048		OPACR16	OPACR17(2)	Reserved					OPACR23
0x004C		OPACR24	Reserved	OPACR26	Reserved				OPACR31
0x0050		OPACR32	Reserved		OPACR35	Reserved		OPACR38	OPACR39
0x0054		Reserved							
0x0058		OPACR48	OPACR49	Reserved					
0x005C		Reserved		OPACR58	OPACR59 (2)	OPACR60(2)	Reserved	OPACR62	Reserved
0x0060		Reserved		OPACR66	OPACR67	OPACR68	OPACR69	Reserved	
0x0064		Reserved							
0x0068		Reserved						OPACR86	OPACR87
0x006C		OPACR88	OPACR89	OPACR90	Reserved	OPACR92	Reserved		

1. Not present in PBRIDGE_1
2. Not present in PBRIDGE_0

13.4.3 Register descriptions

Master Protection Registers (MPROT)

Each MPROT register contains one or more 4-bit fields, called MPROT*n*, as shown in *Table 124*. Each of these fields defines the access privilege level associated with bus master *n* in the platform. The registers provide one field per bus master. See *Section 14.1.4, Logical master IDs*, for a list of master numbers and names.

Each MPROT*n* field has the structure described in *Figure 135* and *Table 126*.

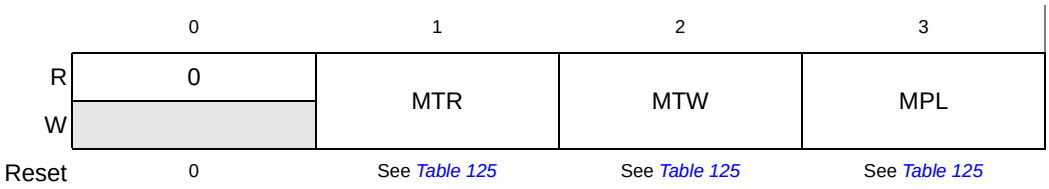


Figure 135. MPROT*n* field structure

Table 125. MPROT*n* field reset values

n	MTR	MTW	MPL
0	1	1	1
1	1	1	1
2	0	0	1
8	1	1	1
9	1	1	1
All others	0	0	0

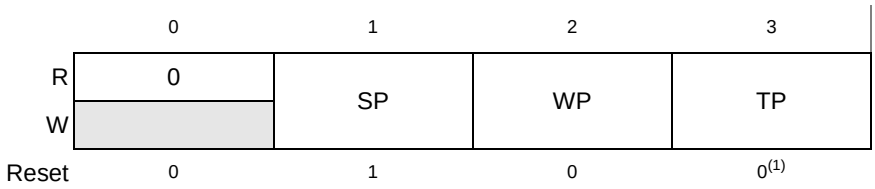
Table 126. MPROT*n* field structure descriptions

Subfield	Description
MTR	Master Trusted for Reads. This bit determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
MTW	Master Trusted for Writes. This bit determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
MPL	Master Privilege Level. This bit determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode. 1 Accesses from this master are not forced to user-mode.

Peripheral access control registers (PACR)

Each PACR register contains one or more 4-bit fields, called PACR*n*, as shown in *Table 124*. Each of these fields defines the access levels supported by the associated module. The lists of modules and their corresponding numbers are shown in *Table 128*.

Each PACR*n* field has the structure described in *Figure 136* and *Table 127*.



1. The reset state of PACR0[TP] is 1, not 0. PACR0[SP] and PACR0[TP] are hard wired to 1 and cannot be changed. Therefore, untrusted masters or user-mode accesses are always denied.

Figure 136. PACR*n* field structure

Table 127. PACR*n* field structure descriptions

Subfield	Description
SP	Supervisor Protect. This bit determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The MPROTx[MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
WP	Write Protect. This bit determines whether the peripheral allows write accesses. 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
TP	Trusted Protect. This bit determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.

Table 128. On-platform peripherals and PACR numbers

Peripheral	PACR number
PBRIDGE	0
XBAR	1(1)
MPU	4
SEMA4	9
SWT	14
STM	15
ECSM	16
eDMA	17(1)

Table 128. On-platform peripherals and PACR numbers (continued)

Peripheral	PACR number
INTC	18

1. Not present in PBRIDGE_1

Off-platform peripheral access control registers (OPACR)

The OPACR defines the access levels supported by the associated module. Each OPACR has a format identical to the PACR described in *Section , Peripheral access control registers (PACR).*

The lists of off-platform modules and their corresponding numbers are listed in *Table 129*.

Table 129. Off-platform peripherals and OPACR numbers

Peripheral[1]	OPACR number
DSPI 0	4
DSPI 1	5
DSPI 2	6
DSPI 3	7
DSPI 4	8
FlexCAN 0	16
FlexCAN 1	17
DMA_MUX	23
FlexRay controller	24
Safety_port	26
BAM	31
ADC0	32
CTU	35
eTimer 0	38
eTimer 1	39
LINFlex 0	48
LINFlex 1	49
CRC	58
FCCU	59
CRC1	60
CFLASH0	66
DFLASH0	67
SIUL	68
WKPU	69
SSCM	86

Table 129. Off-platform peripherals and OPACR numbers (continued)

Peripheral[1]	OPACR number
MC_ME	87
MC_CGM	88
MC_RGM	89
MC_PCU	90
PIT	92

1. All these peripherals are mapped on PBRIDGE_0 except DSPI4, CRC1, FCCU, and FLEXCAN1 which are mapped on PBRIDGE_1

13.5 Functional description

The PBRIDGE serves as an interface between a system bus and the peripheral (slave) bus. It functions as a protocol translator. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

There is no need to configure the PBRIDGE registers unless the default master and/or peripheral access privileges need to be changed.

13.5.1 Access support

Aligned 32-bit word accesses, halfword accesses, and byte accesses are supported for the peripherals. Peripheral registers must not be misaligned, although no explicit checking is performed by the PBRIDGE.

Note: Data accesses that cross a 32-bit boundary are not supported.

Peripheral write buffering

Buffered writes are not supported by the PBRIDGE.

Read cycles

Two-clock read accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary.

Write cycles

Three clock write accesses are possible with the PBRIDGE when the requested access size is 32-bits or smaller. Misaligned writes that cross a 32-bit boundary are not supported.

13.5.2 General operation

Slave peripherals are modules that contain readable/writable control and status registers. The system bus master reads and writes these registers through the PBRIDGE. The PBRIDGE generates module enables, the module address, transfer attributes, byte enables, and write data as inputs to the slave peripherals. The PBRIDGE captures read data from the slave interface and drives it on the system bus.

The PBRIDGE occupies a 64 MB portion of the address space. The register maps of the slave peripherals are located on 16-KB boundaries. Each slave peripheral is allocated one

16-KB block of the memory map, and is activated by one of the module enables from the PBRIDGE.

The PBRIDGE is responsible for indicating to slave peripherals if an access is in supervisor or user mode. All eDMA and FlexRay transfers are done in supervisor mode.

14 Crossbar Switch (XBAR)

14.1 Information specific to this device

This section presents device-specific parameterization, customization, and feature availability information not specifically referenced in the remainder of this chapter.

14.1.1 Register availability

Not all registers listed in *Table 133* are available on this device. Specifically, this device includes only registers for:

- Slaves 0, 1, 2, 3, 6, and 7
- Masters 0, 1, 2, 3, 5, and 6

14.1.2 MPR reset value

The reset value of the MPR register on this device is 0x0540_3210.

14.1.3 max_halt_request signal value

The max_halt_request signal is hardwired to '0' on this device. Halt mode feature is not supported.

14.1.4 Logical master IDs

Table 130 defines the logical master ID used for the chip masters.

Table 130. Logical master IDs

Master	Logical Master ID
Core_0 Instruction port Core_0 Load/Store port	0
Core_1 Instruction port Core_1 Load/Store port	1
eDMA_0	2(1)
Core_0 Nexus	8(1)
Core_1 Nexus	9(1)
FlexRay	3

1. The MPU cannot differentiate between core and Nexus accesses to slave ports.

14.1.5 Master port allocation

Table 131 defines the XBAR master port allocation for this device.

Table 131. XBAR master port allocation

XBAR master port	XBAR module
M0	Core 0 e200z0h complex Instruction port
M1	Core 0 e200z0h complex Load/Store port + Nexus port
M2	DMA_0
M3	FlexRay
M4	—
M5	Core 1 e200z0h complex Instruction port
M6	Core 1 e200z0h complex Load/Store port + Nexus port
M7	—

14.1.6 Slave port allocation

Table 132 defines the XBAR slave port allocation for this device.

Table 132. XBAR slave port allocation

XBAR slave port	XBAR module
S0	Flash Controller Port 0
S1	Flash Controller Port 1
S2	SRAM Controller 0
S3	SRAM Controller 1
S4	—
S5	—
S6	PBRIDGE_1
S7	PBRIDGE_0

14.2 Introduction

14.2.1 Overview

This section provides an overview of the generic multi-layer AHB crossbar switch (XBAR[g]). The purpose of the XBAR is to concurrently support up to eight simultaneous connections between master ports and slave ports. The XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

g. An alternate abbreviation for this module is MAX.

14.2.2 Features

The XBAR has the ability to gain control of all the slave ports and prevent any masters from making accesses to the slave ports. This feature is useful when the user wishes to turn off the clocks to the system and needs to ensure that no bus activity will be interrupted.

The XBAR can put each slave port into a low power park mode so that slave port will not dissipate any power transitioning address, control or data signals when not being actively accessed by a master port.

Each slave port can also support multiple master priority schemes. Each slave port has a hardware input which selects the master priority scheme so the user can dynamically change master priority levels on a slave port by slave port basis.

The XBAR will allow for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will stalled until the higher priority master completes its transactions.

14.2.3 Limitations

The XBAR routes bus transactions initiated on the master ports to the appropriate slave ports. There is no provision included to route transactions initiated on the slave ports to other slave ports or to master ports. Simply put, the slave ports do not support the bus request/bus grant protocol, the XBAR assumes it is the sole master of each slave port.

Since the XBAR does not support the bus request/bus grant protocol, if multiple masters are to be connected to a single master port an external arbiter will need to be used. In the case of a single master connecting to a master port the single master’s bus grant signal must be tied off in the asserted state.

Each master and slave port is fully AHB-Lite + AMBA V6 extensions compliant. The ports are not fully AHB compliant because the XBAR does not support SPLITs or RETRYs.

14.2.4 General Operation

When a master makes an access to the XBAR the access will be immediately taken by the XBAR. If the targeted slave port of the access is available then the access will be immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR. If the targeted slave port of the access is busy or parked on a different master port the requesting master will simply see wait states inserted (**hready** held negated) until the targeted slave port can service the master’s request. The latency in servicing the request will depend on each master’s priority level and the responding peripheral’s access time.

Since the XBAR appears to be just another slave to the master device, the master device will have no knowledge of whether or not it actually owns the slave port it is targeting. While the master does not have control of the slave port it is targeting it will simply be wait stated.

A master will be given control of the targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has an outstanding request to one slave port that has a long response time, has a pending access to a different slave port, and

a lower priority master is also making a request to the same slave port as the pending access of the higher priority master.

Once the master has control of the slave port it is targeting the master will remain in control of that slave port until it gives up the slave port by running an IDLE cycle or by leaving that slave port for its next access. The master could also lose control of the slave port if another higher priority master makes a request to the slave port; however, if the master is running a locked or fixed length burst transfer it will retain control of the slave port until that transfer is completed. Based on the AULB bit in the MGPCR (Master General Purpose Control Register) the master will either retain control of the slave port when doing undefined length incrementing burst transfers or will lose the bus to a higher priority master.

The XBAR will terminate all master IDLE transfers (as opposed to allowing the termination to come from one of the slave busses). Additionally, when no master is requesting access to a slave port the XBAR will drive IDLE transfers onto the slave bus, even though a default master may be granted access to the slave port. When the XBAR is controlling the slave bus (that is, during low power park) the **hmaster** field will indicate 4'b0000.

When a slave bus is being IDLEd by the XBAR it can park the slave port on the master port indicated by the PARK bits in the SGPCR (Slave General Purpose Control Register). This can be done in an attempt to save the initial clock of arbitration delay that would otherwise be seen if the master had to arbitrate to gain control of the slave port. The slave port can also be put into low power park mode in attempt to save power.

14.3 XBAR registers

This section provides information on XBAR registers.

14.3.1 Register summary

There are two registers that reside in each slave port of the XBAR and one register that resides in each master port of the XBAR. These registers are IP bus compliant registers. Read and write transfers both require two IP bus clock cycles. The registers can only be read from and written to in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses.

The registers are fully decoded and an error response is returned if an unimplemented location is accessed within the XBAR.

The slave registers also feature a bit, which when written with a 1, will prevent the registers from being written to again. The registers will still be readable, but future write attempts will have no effect on the registers and will be terminated with an error response.

The memory map for the XBAR program-visible registers is shown in *Table 133*. *Table 134* shows the XBAR register summary.

Table 133. XBAR register configuration summary

Offset from XBAR_BASE (0xFFF0_4000)	Register	Use
0x000	MPR0	Master Priority Register for Slave port 0
0x010	SGPCR0	General Purpose Control Register for Slave port 0
0x100	MPR1	Master Priority Register for Slave port 1

Table 133. XBAR register configuration summary (continued)

Offset from XBAR_BASE (0xFFF0_4000)	Register	Use
0x110	SGPCR1	General Purpose Control Register for Slave port 1
0x200	MPR2	Master Priority Register for Slave port 2
0x210	SGPCR2	General Purpose Control Register for Slave port 2
0x300	MPR3	Master Priority Register for Slave port 3
0x310	SGPCR3	General Purpose Control Register for Slave port 3
0x400	MPR4	Master Priority Register for Slave port 4
0x410	SGPCR4	General Purpose Control Register for Slave port 4
0x500	MPR5	Master Priority Register for Slave port 5
0x510	SGPCR5	General Purpose Control Register for Slave port 5
0x600	MPR6	Master Priority Register for Slave port 6
0x610	SGPCR6	General Purpose Control Register for Slave port 6
0x700	MPR7	Master Priority Register for Slave port 7
0x710	SGPCR7	General Purpose Control Register for Slave port 7
0x800	MGPCR0	General Purpose Control Register for Master port 0
0x900	MGPCR1	General Purpose Control Register for Master port 1
0xA00	MGPCR2	General Purpose Control Register for Master port 2
0xB00	MGPCR3	General Purpose Control Register for Master port 3
0xC00	MGPCR4	General Purpose Control Register for Master port 4
0xD00	MGPCR5	General Purpose Control Register for Master port 5
0xE00	MGPCR6	General Purpose Control Register for Master port 6
0xF00	MGPCR7	General Purpose Control Register for Master port 7

Figure 137. Key to Register Fields

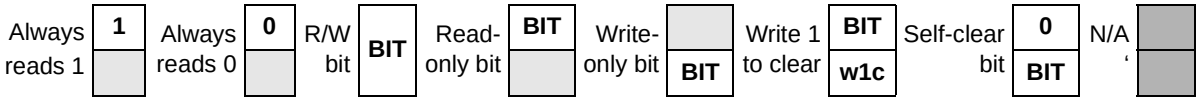


Table 134. XBAR register summary

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MPRn ($BASE + 0x000 + n*0x100)	R	0	MSTR_7			0	MSTR_6			0	MSTR_5			0	MSTR_4		
	W																
	R	0	MSTR_3			0	MSTR_2			0	MSTR_1			0	MSTR_0		
	W																
SGPCRn ($BASE + 0x010 + n*0x100)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	RO	HLP							HPE7	HPE6	HPE5	HPE4	HPE3	HPE2	HPE1	HPE0

Table 134. XBAR register summary (continued)

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W							ARB				PCTL			PARK		
MGPCRn ($BASE + 0x800 + n*0x100	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W														AULB		

Note: for n = 0 to 7

14.3.2 XBAR register descriptions

The following paragraphs provide detailed descriptions of the various XBAR registers.

Table 135 provides a key to the terms found in XBAR registers.

Table 135. Register terms

Term	Description
Gray bit	Unimplemented bit; always reads as zero;writing has no effect
Access	
S	Supervisor mode only
-	Supervisor or user mode
Type	
r	Read only; writing to this bit has no effect
w	Write only
rw	Standard read/write bit. Only software can change a bit's value (other than a hardware reset).
rwm	A read/write bit that may be modified by hardware in some fashion other than reset.
w1c	A status bit that can be read and cleared by writing a logic 1
slfclr	Self-clearing bit. Writing a 1 has some effect on module, but it always reads as a 0.
Reset	
0	Resets to a logic 0
1	Resets to a logic 1
u	Unaffected by reset
?	Reset state is unknown.

Master Priority Register

The Master Priority Register (MPR) sets the priority of each master port on a per slave port basis and resides in each slave port.

MPRn — **Master Priority Register n** — **Addr $BASE + 0x000 + n*100, Wait State: 0, Access: S**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		MSTR_7				MSTR_6				MSTR_5				MSTR_4		
TYPE:	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw
RESET:	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
		MSTR_3				MSTR_2				MSTR_1				MSTR_0		
TYPE:	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw
RESET:	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Note: for n = 0 to 7

Figure 138. Master Priority Register n

Table 136. Master Priority Register descriptions

Name	Description	Settings
Bit 0	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_7	**Master 7 Priority -** These bits set the arbitration priority for master port 7 on the associated slave port. These bits are initialized by hardware reset. The reset value is 000	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 4	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_6	**Master 6 Priority -** These bits set the arbitration priority for master port 6 on the associated slave port. These bits are initialized by hardware reset. The reset value is 101	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 8	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA

Table 136. Master Priority Register descriptions (continued)

Name	Description	Settings
MSTR_5	**Master 5 Priority -** These bits set the arbitration priority for master port 5 on the associated slave port. These bits are initialized by hardware reset. The reset value is 100	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 12	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_4	**Master 4 Priority -** These bits set the arbitration priority for master port 4 on the associated slave port. These bits are initialized by hardware reset. The reset value is 000	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 16	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_3	**Master 3 Priority -** These bits set the arbitration priority for master port 3 on the associated slave port. These bits are initialized by hardware reset. The reset value is 011	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 20	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_2	**Master 2 Priority -** These bits set the arbitration priority for master port 2 on the associated slave port. These bits are initialized by hardware reset. The reset value is 010	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 24	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_1	**Master 1 Priority -** These bits set the arbitration priority for master port 1 on the associated slave port. These bits are initialized by hardware reset. The reset value is 001	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.
Bit 28	**Master Priority Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
MSTR_0	**Master 0 Priority -** These bits set the arbitration priority for master port 0 on the associated slave port. These bits are initialized by hardware reset. The reset value is 000	000This master has the highest priority when accessing the slave port. 111This master has the lowest priority when accessing the slave port.

The Master Priority Register can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the slave General Purpose Control Register the Master Priority Register can only be read from, attempts to write to it will have no effect on the MPR and result in an error response.

Note: No two available master ports may be programmed with the same priority level. Attempts to program two or more available masters with the same priority level will result in an error response and the MPR will not be updated.

Slave General Purpose Control Register

The Slave General Purpose Control Register (SGPCR) controls several features of each slave port.

The Read Only (RO) bit will prevent any registers associated with this slave port from being written to once set. This bit may be written with 0 as many times as the user desires, but once it is written to a 1 only a reset condition will allow it to be written again.

The Halt Low Priority (HLP) bit has no effect since max_halt_request input is hard wired to zero for SPC56xP60/54.

The PCTL bits determine how the slave port will park when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low power park mode which will force all the outputs of the slave port to inactive states when no master is requesting an access. The low power park feature can result in an overall power savings if a the slave port is not saturated; however, it will force an extra clock of latency whenever any master tries to access it when it is not in use because it will not be parked on any master.

The PARK bits determine which master the slave will park on when no master is making an active request. Please use caution to only select master ports that are actually present in the design. If the user programs the PARK bits to a master not present in the current design implementation undefined behavior will result.

SGPCRn	Slave General Purpose Control Register n	Addr $BASE + 0x010 + n*100 Wait State: 0 Access: S

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	RO	HLP							HPE 7	HPE 6	HPE 5	HPE 4	HPE 3	HPE 2	HPE 1	HPE 0
TYPE:	rw	rw	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Once the RO bit is written to a 1, only hardware reset will return it to a 0.

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
							ARB				PCTL			PARK		
TYPE:	r	r	r	r	r	r	rw	rw	r	r	rw	rw	r	rw	rw	rw
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Note: for n = 0 to 7

Figure 139. Slave General Purpose Control Register n

Table 137. Slave General Purpose Control Register Descriptions

Name	Description	Setting
RO	**Read Only -** This bit is used to force all of a slave port's registers to be read only. Once written to 1 it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0	0All this slave port's registers can be written. 1All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
HLP	**Halt Low Priority -** This bit is used to set the initial arbitration priority of the max_halt_request input. This bit is initialized by hardware reset. The reset value is 0	0The max_halt_request input has the highest priority for arbitration on this slave port 1The max_halt_request input has the lowest initial priority for arbitration on this slave port.
Bits 2–7	**Slave General Purpose Control Register Reserved -** These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.	NA
HPEx	**High Priority Enable -** These bits are used to enable the mX_high_priority inputs for the respective master. These bits are initialized by hardware reset. The reset value is 0	0The mX_high_priority input is disabled on this slave port 1The mX_high_priority input is enabled on this slave port.

Table 137. Slave General Purpose Control Register Descriptions (continued)

Name	Description	Setting
Bits 16–21	**Slave General Purpose Control Register Reserved -** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.	NA
ARB	**Arbitration Mode -** These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 00	00Fixed Priority. 01Round Robin (rotating) Priority. 10Reserved 11Reserved
Bits 24–25	**Slave General Purpose Control Register Reserved -** These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.	NA
PCTL	**Parking Control -** These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 00.	00When no master is making a request the arbiter will park the slave port on the master port defined by the PARK bit field. 01When no master is making a request the arbiter will park the slave port on the last master to be in control of the slave port. 10When no master is making a request the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11Reserved
Bit 28	**Slave General Purpose Control Register Reserved -** This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.	NA
PARK	**PARK -** These bits are used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset. The reset value is 000	000Park on Master Port 0 001Park on Master Port 1 010Park on Master Port 2 011Park on Master Port 3 100Park on Master Port 4 101Park on Master Port 5 110Park on Master Port 6 111Park on Master Port 7

The SGPCR can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the SGPCR the SGPCR can only be read, attempts to write to it will have no effect on the SGPCR and result in an error response.

Master General Purpose Control Register

The Master General Purpose Control Register (MGPCR) presently controls only whether or not the master's undefined length burst accesses will be allowed to complete uninterrupted or whether they can be broken by requests from higher priority masters.

The AULB (Arbitrate on Undefined Length Bursts) bit field determines whether (and when) or not the XBAR will arbitrate away the slave port the master owns when the master is performing undefined length burst accesses.

If the user has configured the XBAR to have less than eight master ports only the registers associated with the remaining master ports will be present, all other registers will become reserved locations in memory.

MGPCRn **Master General Purpose Control Register n**

Addr
$BASE + 0x800 + n*100
Wait State: 0
Access: S

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TYPE:	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
														AULB		
TYPE:	r	r	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Note: for n = 0 to 7

Figure 140. Master General Purpose Control Register n

Table 138. Master General Purpose Control Register Descriptions

Name	Description	Setting
Bits 0–28	**Master General Purpose Control Register Reserved -** These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.	NA
AULB	**Arbitrate on Undefined Length Bursts -** These bits are used to select the arbitration policy during undefined length bursts by this master. These bits are initialized by hardware reset. The reset value is 000	000No arbitration will be allowed during an undefined length burst. 001Arbitration will be allowed at any time during an undefined length burst. 010Arbitration will be allowed after four beats of an undefined length burst. 011Arbitration will be allowed after eight beats of an undefined length burst. 100Arbitration will be allowed after 16 beats of an undefined length burst. 101Reserved 110Reserved 111Reserved

The MGPCR can only be accessed in supervisor mode with 32-bit accesses.

14.3.3 Coherency

Since the content of the registers has a real time effect on the operation of the XBAR it is important for the user to understand that any register modifications take effect as soon as the register is written. The values of the registers do not track with slave port related AHB accesses but instead track only with IP bus accesses.

The exception to this rule are the AULB bits in the MGPCR. These update of these bits is only recognized when the master on that master port runs an IDLE cycle, even though the IP bus cycle to write them will have long since terminated successfully. If the AULB bits in the MGPCR are written in between two burst accesses the new AULB encodings will not take effect until an IDLE cycle has been initiated by the master on that master port.

14.4 Function

This section describes in more detail the functionality of the XBAR.

14.4.1 Arbitration

The XBAR supports two arbitration schemes; a simple fixed-priority comparison algorithm, and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

Arbitration During Undefined Length Bursts

Arbitration points during an undefined length burst are defined by the current master's MGPCR AULB field setting. When a defined length is imposed on the burst via the AULB bits the undefined length burst will be treated as a single or series of single back to back fixed length burst accesses.

Example: A master runs an undefined length burst and the AULB bits in the MGPCR indicate arbitration will occur after the fourth beat of the burst. The master runs two sequential beats and then starts what will be an 12 beat undefined length burst access to a new address within the same slave port region as the previous access. The XBAR will not allow an arbitration point until the fourth overall access (second beat of the second burst). At that point all remaining accesses will be open for arbitration until the master loses control of the slave port.

Assume the master loses control of the slave port after the fifth beat of the second burst. Once the master regains control of the slave port no arbitration point will be available until after the master has run four more beats of its burst. After the fourth beat of the (now continued) burst (ninth beat of the second burst from the master's perspective) is taken all beats of the burst will once again be open for arbitration until the master loses control of the slave port.

Assume the master again loses control of the slave port on the fifth beat of the third (now continued) burst (10th beat of the second burst from the master's perspective). Once the master regains control of the slave port it will be allowed to complete its final two beats of its burst without facing arbitration.

Note that fixed length burst accesses are not affected by the AULB bits. All fixed length burst accesses lock out arbitration until the last beat of the fixed length burst.

Fixed Priority Operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the MPR (Master Priority Register). If two masters both request access to a slave port the master with the highest priority in the selected priority register will gain control over the slave port.

Any time a master makes a request to a slave port the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port the new requesting master will be granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed length burst transfer or a locked transfer. In this case the new requesting master will have to wait until the end of the burst transfer or locked transfer before it will be granted control of the slave port. If the master is running an undefined length burst transfer the new requesting master must wait until an arbitration point for the undefined length burst transfer before it will be granted control of the slave port. Arbitration points for an undefined length burst are defined in the MGPCR for each master.

If the new requesting master's priority level is lower than that of the master that currently has control of the slave port the new requesting master will be forced to wait until the master that currently has control of the slave port either runs an IDLE cycle or runs a non IDLE cycle to a location other than the current slave port.

Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master number.This relative priority is compared to the ID of the last master to perform a transfer on the slave bus. The highest priority requesting master will become owner of the slave bus as the next transfer boundary (accounting for locked and fixed-length burst transfers). Priority is based on how far ahead the ID of the requesting master is to the ID of the last master (ID is defined by master port number, not the **hmaster** field).

Once granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line will be granted access to the slave port at the next assertion of **sX_hready**, or possibly on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the XBAR is implemented with master ports 0, 1, 4 and 5. If the last master of the slave port was master 1, and master 0, 4 and 5 make simultaneous requests, they will be serviced in the order 4, 5 and then 0.

Parking may still be used in a round-robin mode, but will not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff will occur to the next master in line after one cycle of arbitration. If the slave port is put into low power park mode the round-robin pointer will be reset to point at master port 0, giving it the highest priority.

Each master port has an **mX_high_priority** input which can be enabled by writing the correct data to the SGPCR. If a master's **mX_high_priority** input is enabled for a slave port programmed for round-robin mode, that master can force the slave port into fixed priority mode by asserting its **mX_high_priority** input while making a request to that particular slave port. While that (or any enabled) master's **mX_high_priority** input is asserted while making an access attempt to that particular slave port, the slave port will remain in fixed

priority mode. Once that (or any enabled) master's **mX_high_priority** input is negated, or the master no longer attempts to make accesses to that particular slave port, the slave port will revert back to round-robin priority mode and the pointer will be set on the last master to access the slave port.

14.4.2 Priority Assignment

Each master port needs to be assigned a unique 3-Bit priority level. If an attempt is made to program multiple master ports with the same priority level within a register (MPR) the XBAR will respond with an error and the registers will not be updated.

Priority Elevation

The XBAR has a hardware input per master port (**mX_high_priority**) which is used to temporarily elevate the master's priority level on all slave ports. When the master's **mX_high_priority** input is asserted the master port will automatically have higher priority than all other master ports that do not have their **mX_high_priority** input asserted regardless of the priority levels programmed in the MPR and AMPR. If multiple master ports have their **mX_high_priority** input asserted they will have higher priority than all master ports which do not have their **mX_high_priority** inputs asserted. The MPR priority level will determine which master port that has its **mX_high_priority** input asserted has the highest priority on a slave port by slave port basis.

This functionality is useful because it allows the user to automatically elevate a master port's priority level throughout the XBAR in order to quickly perform temporary tasks such as servicing interrupts.

Please note that the HPEx bits must be set in the SGPCR in the slave port in order for the **mX_high_priority** inputs to be received by the slave port.

14.4.3 Master Port Functionality

General

Each master port consists of two decoders, a capture unit, a register slice, a mux and a small state machine.

The first decoder is used to decode the **mX_hsel_slv** and control signals coming directly from the master, telling the state machine where the master's next access will be and if it is in fact a legal access. The second decoder receives its input from the capture unit, so it may be looking directly at the signals coming from the master or it may be looking at captured signals coming from the master, depending entirely on the state of the targeted slave port. The second decoder is then used to generate the access requests that go to the slave ports.

The capture unit is used to capture the address and control information coming from the master in the event that the targeted slave port cannot immediately service the master. The capture unit is controlled by outputs from the state machine which tell it to either pass through the original master signals or the captured signals.

The register slice contains the registers associated with the specific master port. The registers have a quasi-IP bus interface at this level for reads and writes and the outputs feed directly into the state machine.

The mux is used simply to select which slave's read data is sent back to the master. The mux is controlled by the state machine.

The state machine controls all aspects of the master port. It knows which slave port the master wants to make a request to and controls when that request is made. It also has knowledge of each slave port, knowing whether or not the slave port is ready to accept an access from the master port. This will determine whether or not the master may immediately have its request taken by the slave port or whether the master port will have to capture the master's request and queue it at the slave port boundary.

Master Port Decoders

The decoders are very simple as they ensure an access request is allowed to be made and that the slave port targeted is actually present in the design. The decoders feeding the state machine are always enabled. The decoders that select the slave are enabled only when the master port controlling state machine wants to make a request to a slave port. This is necessary so that if a master port is making an access to a slave port and is being wait stated, and its next access is to a different slave port, the request to the second slave port can be held off until the access to the first slave port is terminated.

The decoders also output a "hole decode" or illegal access signal which tells the state machine that the master is trying to access a slave port that does not exist.

Master Port Capture Unit

The capture unit simply captures the state of the master's address and control signals if the XBAR cannot immediately pass the master's request through to the proper slave port. The capture unit consists of a set of flops and a mux which selects either the asynchronous path from address and control or the flopped (captured) address and control information.

Master Port Registers

The registers in the master port are only those registers associated with this particular master port. The read and write interface for the registers is a quasi-IP bus interface. It is not a full IP bus interface at this level because not all the IP bus signals are routed this deep in the design.

There is a register control block at the same level of the master port and slave port instantiations in the XBAR. This control block ensures that all accesses are 32-bit supervisor accesses before passing them on to the master ports.

The register outputs are connected directly to the state machine.

Master Port State Machine

Master Port State Machine States

The master side state machine's main function is to monitor the activities of the master port. The state machine has six states: **busy**, **idle**, **waiting**, **stalled**, **steady state**, **first cycle error response** and **second cycle error response**.

The **busy** state is used when the master runs a BUSY cycle to the master port. The master port maintains its request to the slave port if it currently owns the slave port; however, if it loses control of the slave port it will no longer maintain its request. If the master port loses control of the slave port it will not be allowed to make another request to the slave port until it runs a NSEQ or SEQ cycle.

The **idle** state is used when the master runs a valid IDLE cycle to the master port. The master port makes no requests to the slave ports (disables the slave port decoder) and terminates the IDLE cycle.

The **waiting** state is used when the **hsel** signal is negated to the master port, indicating the master is running valid cycles to a local slave other than the XBAR. In this case the max disables the slave port decoder and holds **hresp** and **hready** negated.

The **stalled** state is used when the master makes a request to a slave port that is not immediately ready to receive the request. In this case the state machine will direct the capture unit to send out the captured address and control signals and will enable the slave port decoder to indicate a pending request to the appropriate slave port.

The **steady state** state is used when the master port and slave port are in fully asynchronous mode, making the XBAR completely transparent in the access. The state machine selects the appropriate slave's **hresp, hready** and **hrdata** to pass back to the master.

The **first cycle error response** and **second cycle error response** states are self explanatory. The XBAR will respond with an error response to the master if the master tries to access an unimplemented memory location through the XBAR (that is, a slave port that does not exist).

Master Port State Machine Slave Swapping

The design of the master side state machine is fairly straightforward. The one real decision to be made is how to handle the master moving from one slave port access to another slave port access. The approach that was taken is to minimize or eliminate when possible any "bubbles" that would be inserted into the access due to switching slave ports.

The state machine will not allow the master to request access to another slave port until the current access being made is terminated. This prevents a single master from owning two slave ports at the same time (the slave port it is currently accessing and the slave port it wishes to access next).

The state machine also maintains watch on the slave port the master is accessing as well as the slave port the master wishes to switch to. If the new slave port is parked on the master then the master will be able to make the switch without incurring any delays. The termination of the current access will also act as the launch of the new access on the new slave port. If the new slave port is not parked on the master then the master will incur a minimum one clock delay before it can launch its access on the new slave port.

This is the same for switching from the **busy, idle** or **waiting** state to actively accessing a slave port. If the slave port is parked on the master the state machine will go to the **steady state** state and the access will begin immediately. If the slave port is not parked on the master (serving another master, parked on another master or in low power park mode) then the state machine will transition to the **stalled** state and at least a one clock penalty will be paid.

14.4.4 Slave Port Functionality

General

Each slave port consists of a register slice, a bank of muxes and a state machine.

The register slice contains the registers associated with the specific slave port. The registers have a quasi-IP bus interface at this level for reads and writes and the outputs feed directly into the state machine.

The muxes are a series of 8 to 1 muxes that take in all the address, control and write data information from each of the master ports and then pass the correct master's signals to the slave port. The state machine controls all the muxes.

The state machine is where the main slave port arbitration occurs, it decides which master is in control of the slave port and which master will be in control of the slave port in the next bus cycle.

Slave Port Muxes

The XBAR instantiates many 8 to 1 muxes, one for each master-to-slave signal in fact. All the muxes are designed in an AND - OR fashion, so that if no master is selected the output of the muxes will be zero. (This is an important feature for low power park mode.)

The muxes also have an override signal which is used by the slave port to asynchronously force IDLE cycles onto the slave bus. When the state machine forces an IDLE cycle it zeros out **htrans** and **hmastlock**, making sure the slave bus sees a valid IDLE cycle being run by the XBAR.

The enable to the mux controlling **htrans** also contains an additional control signal from the state machine so that a NSEQ transaction can be forced. This is done any time the slave port switches masters to ensure that no IDLE-SEQ, BUSY-SEQ or NSEQ-SEQ transactions are seen on the slave port when they shouldn't be. If the state machine indicates to run both an IDLE and an NSEQ cycle, the IDLE directive will have priority.

Note: IDLE-SEQ is in fact an illegal access, but a possible scenario given the multi-master environment in the XBAR unless corrected by the XBAR.

Slave Port Registers

There is a register control block at the same level of the master port and slave port instantiations in the XBAR. This control block ensures that all accesses are 32-bit supervisor accesses before passing them on to the master and slave ports.

The registers in the slave port are only those registers associated with this particular slave port. The read and write interface for the registers is a quasi-IP bus interface. It is not a full IP bus interface at this level because not all the IP bus signals are routed this deep in the design.

The register outputs are connected directly to the slave state machine. The registers can be read from an unlimited number of times. The registers can only be written to as long as the RO bit is written to 0 in the SGPCR, once it is written to a 1 only a hardware reset will allow the registers to be written again.

Slave port state machine

Slave Port State Machine States

At the heart of the slave port is the state machine. The state machine is simplicity itself, requiring only four states - **steady state**, **transition state, transition hold state** and **hold state**. Either the slave port is owned by the same master it was in the last clock cycle (either by active use or by parking), it is transitioning to a new master (either for active use or parking), it is transitioning to a new master during wait states or it is being held on the same master pending a transition to a new master.

Slave Port State Machine Arbitration

The real work in the state machine is determining which master port will be in control of the slave port in the next clock cycle, the arbitration. Each master is programmed with a fixed 3 bit priority level. A fourth priority bit is derived from the **mX_high_priority** inputs on the master ports, effectively making each master's priority level a 4 bit field with **mX_high_priority** being the MSB. The XBAR uses these bits in determining priority levels when programmed for fixed priority mode of operation or when one of the enabled **mX_high_priority** inputs is asserted.

Arbitration always occurs on a clock edge, but only occurs on edges when a change in mastership will not violate AHB-Lite protocols. Valid arbitrations points include any clock cycle in which **sX_hready** is asserted (provide the master is not performing a burst or locked cycle) and any wait state in which the master owning the bus indicates a transfer type of IDLE (provided the master is not performing a locked cycle).

Since arbitration can occur on every clock cycle the slave port masks off all master requests if the current master is performing a locked transfer or a protected burst transfer, guaranteeing that no matter how low its priority level it will be allowed to finish its locked or protected portion of a burst sequence.

Slave Port State Machine Master Handoff

The only times the slave port will switch masters when programmed for fixed priority mode of operation is when a higher priority master makes a request or when the current master is the highest priority and it gives up the slave port by either running and IDLE cycle to the slave port or running a valid access to a location other than the slave port.

If the current master loses control of the slave port because a higher priority master takes it away, the slave port will not incur any wasted cycles. The current master has its current cycle terminated by the slave port at the same time the new master's address and control information are recognized by the slave port. This appears as a seamless transition on the slave port.

If the current master is being wait-stated when the higher priority master makes its request, then the current master will be allowed to make one more transaction on the slave bus before giving it up to the new master. *Figure 141* illustrates the effect of a higher priority master taking control of the bus when the slave port is programmed for a fixed priority mode of operation.

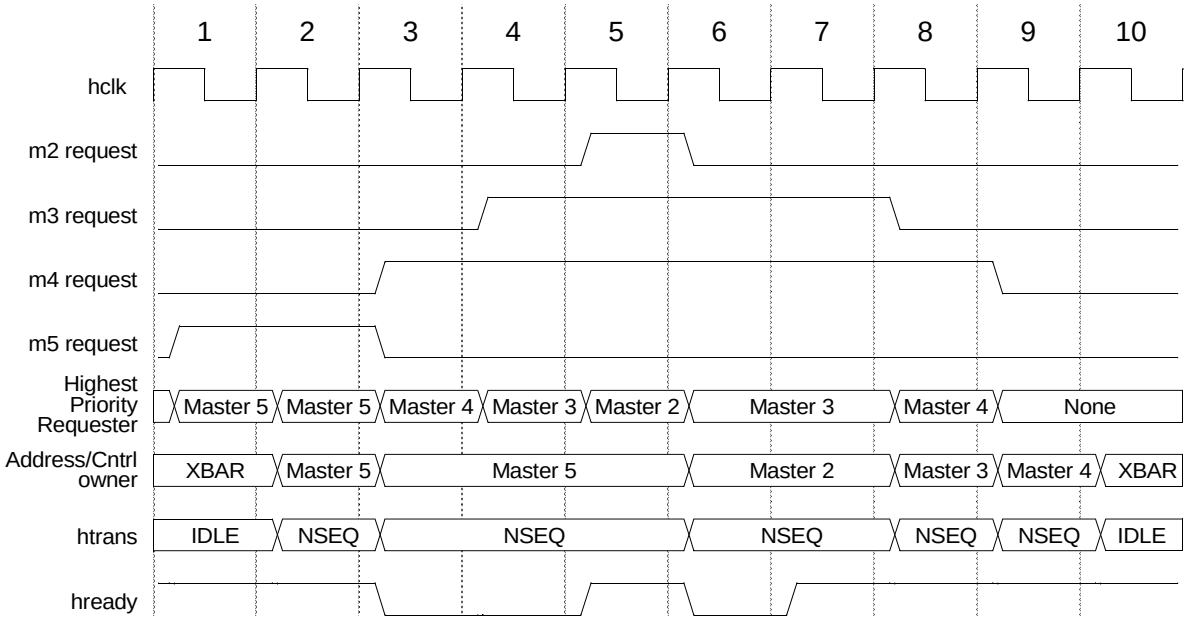


Figure 141. Low to high priority mastership change

If the current master is the highest priority master and it gives up the slave port by running an IDLE cycle or by running a valid cycle to another location other than the slave port the next highest priority master will gain control of the slave port. If the current access incurs any wait states then the transition will be seamless and no bandwidth will be lost; however, if the current transaction is terminated without wait states then one IDLE cycle will be forced onto the slave bus by the XBAR before the new master will be able to take control of the slave port. If no other master is requesting the bus then IDLE cycles will be run by the XBAR but no bandwidth will truly be lost since no master is making a request. *Figure 142* illustrates the effect of a higher priority master giving up control of the bus.

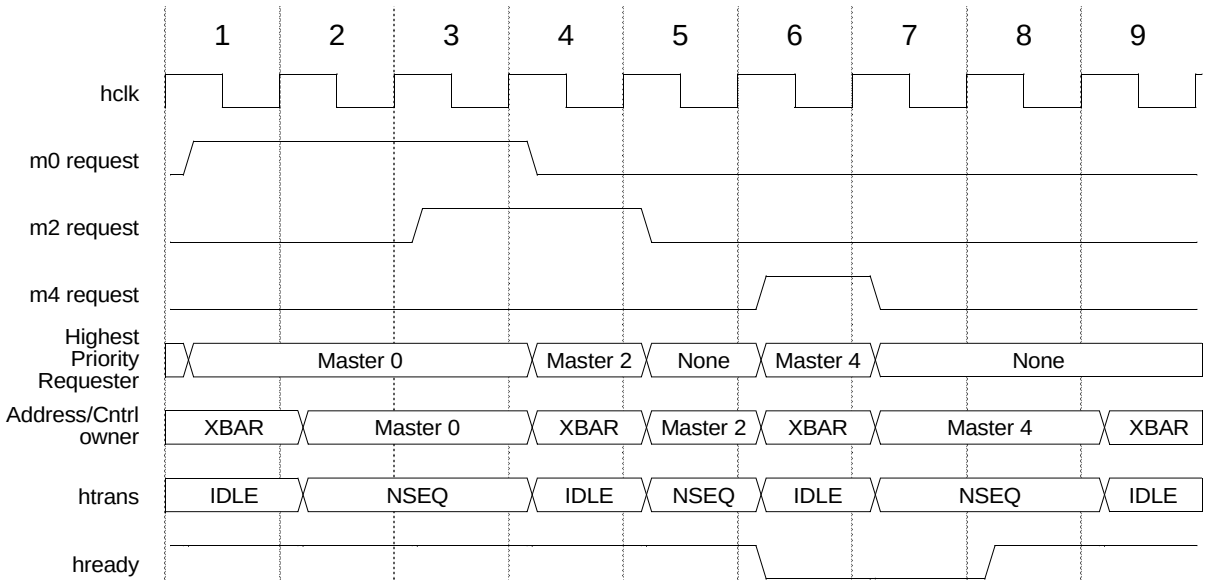


Figure 142. High to low priority mastership change

When the slave port is programmed for round-robin mode of arbitration then the slave port will switch masters any time there is more than one master actively making a request to the slave port. This will happen because any master other than the one which presently owns the bus will be considered to have higher priority. *Figure 143* shows an example of round-robin mode of operation.

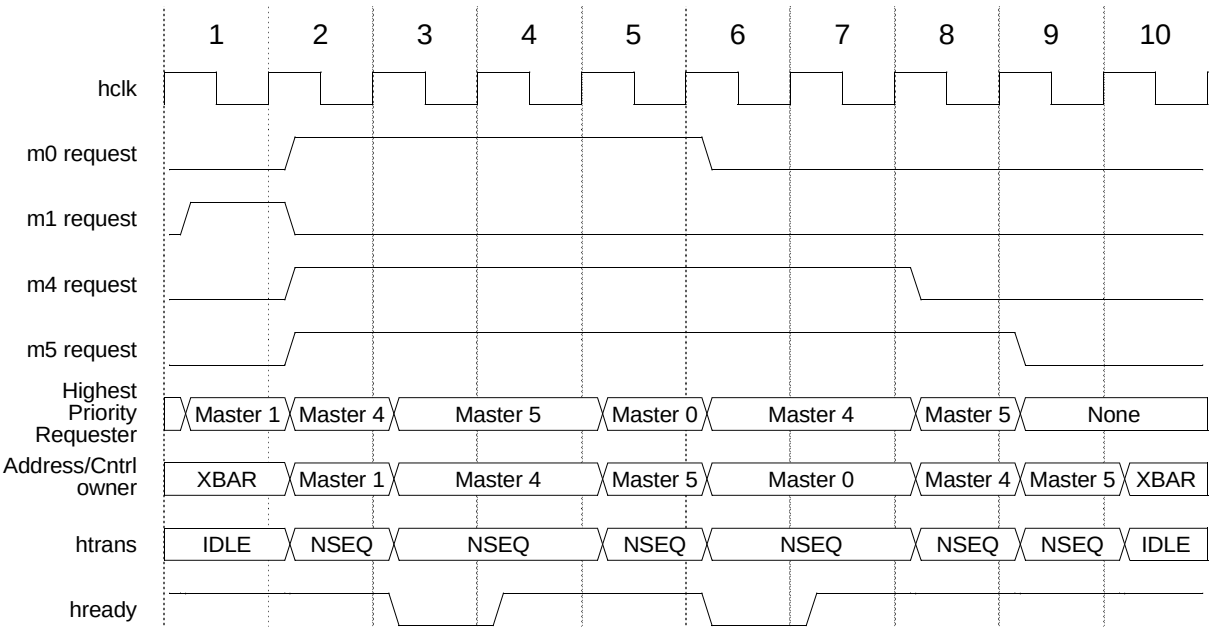


Figure 143. Round-robin mastership change

Slave port state machine parking

If no master is currently making a request to the slave port then the slave port will be parked. It will park in one of four places, dictated by the PCTL and PARK bits in the GPCR and the locked state of the last master to access it.

If the last master to access the slave port ran a locked cycle and continues to run locked cycles even after leaving the slave port the slave port will park on that master without regard to the bit settings in the GPCR and without regard to pending requests from other masters. This is done so a master can run a locked transfer to the slave port, leave it, and return to it and be guaranteed that no other master has had access to it (provided the master maintains all transfers are locked transfers). If locking is not an issue for parking the GPCR bits will dictate the parking method.

If the PCTL bits are set for “low power park” mode then the slave port will enter low power park mode. It will not recognize any master as being in control of it and it will not select any master’s signals to pass through to the slave bus. In this case all slave bus activity will effectively halt because all slave bus signals being driven from the XBAR will be 0. This of course can save quite a bit of power if the slave port will not be in use for some time. The down side is that when a master does make a request to the slave port it will be delayed by one clock since it will have to arbitrate to acquire ownership of the slave port.

If the PCTL bits are set to “park on last” mode then the slave port will park on the last master to access it, passing all that masters signals through to the slave bus. The XBAR will asynchronously force **htrans[1:0]**, **hmaster[3:0]**, **hburst[2:0]** and **hmastlock** to 0 for all access that the master does not run to the slave port. When that master access the slave port again it will not pay any arbitration penalty; however, if any other master wishes to access the slave port a one clock arbitration penalty will be imposed.

If the PCTL bits are set to “use PARK/APARK” mode then the slave port will park on the master designated by the PARK bits. The behavior here is the same as for the “park on last” mode with the exception that a specific master will be parked on instead of the last master to access the slave port. If the master designated by the PARK bits tries to access the slave port it will not pay an arbitration penalty while any other master will pay a one clock penalty.

Figure 144 illustrates parking on a specific master.

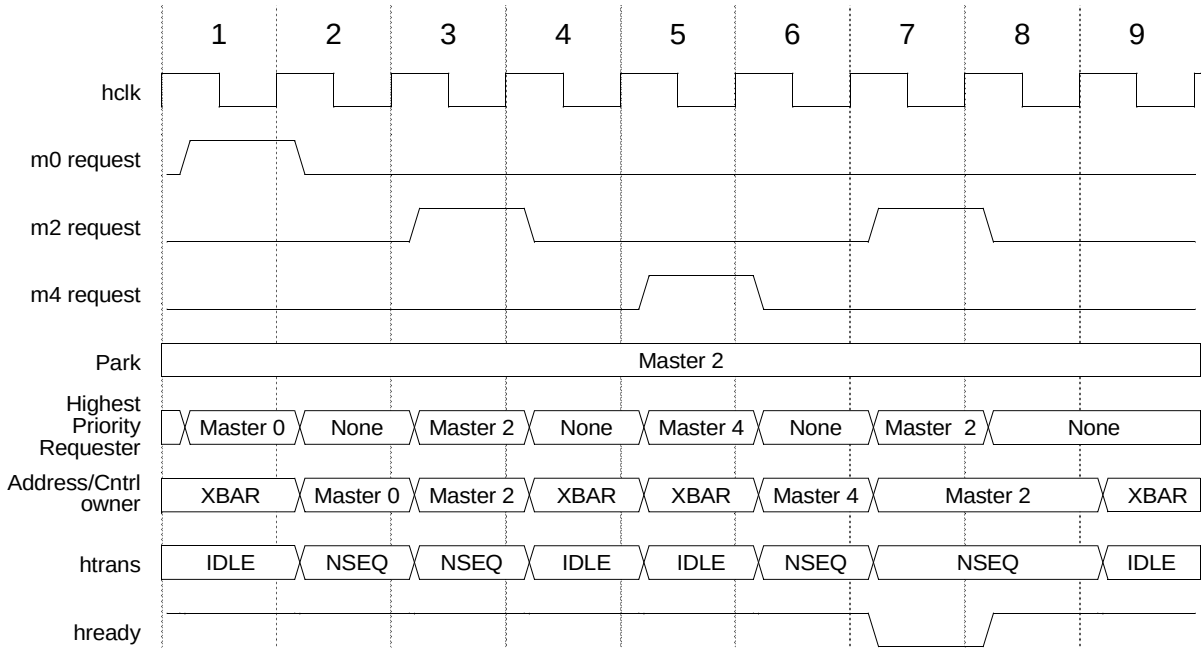


Figure 144. Parking on a specific master

Figure 145 illustrates parking on the last master. Note that in cycle 6 simultaneous requests are made by master 2 and master 4. Although master 2 has higher priority, the slave bus is parked on master 4 so master 4's access will be taken first. The slave port parks on master 2 once it has given control to master 2. This same situation can occur when parking on a specific master as well.

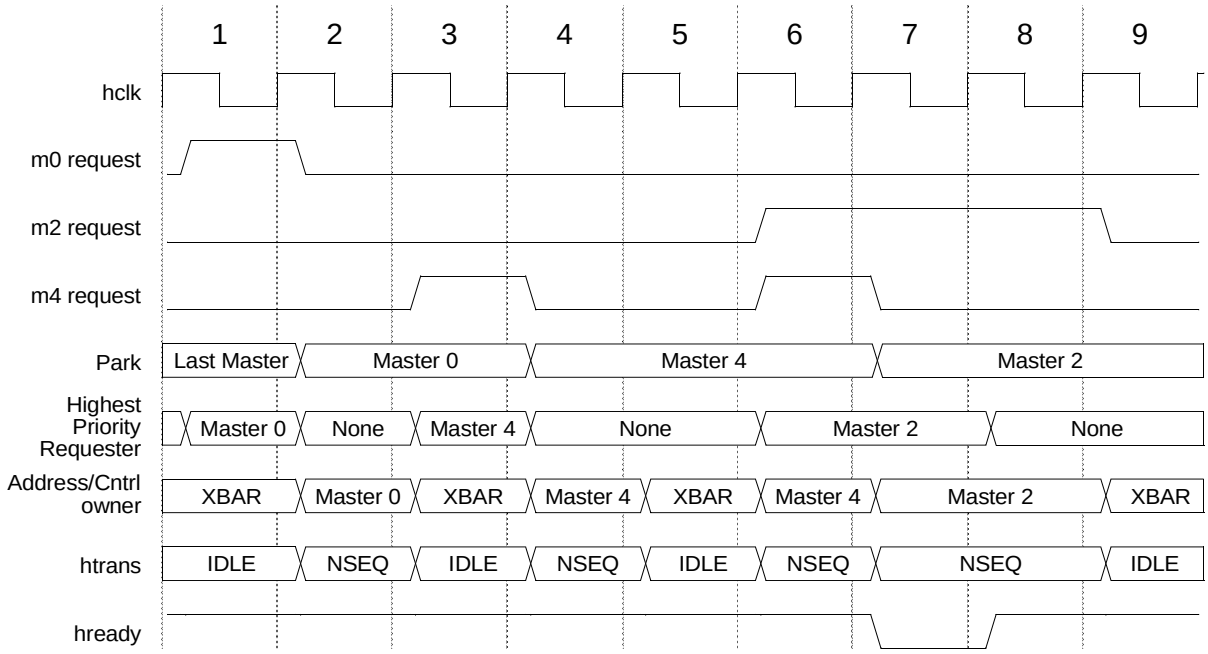


Figure 145. Parking on last master

14.5 Initialization/Application Information

No initialization is required by or for the XBAR. Hardware reset ensures all the register bits used by the XBAR are properly initialized.

14.6 Interface

This section provides information on the XBAR interface.

14.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate in parallel with multiple slaves. In order to maximize data throughput it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR will stall the masters or insert bubbles on the slave side.

14.6.2 Master Ports

Master accesses will receive one of four responses from the XBAR. They will either be ignored, terminated, taken, stalled or responded to with an error.

Ignored Accesses

A master access will be ignored if the **hsel** input of the XBAR is not asserted. The XBAR will respond to IDLE transfers when the **hsel** input is asserted but will not allow the access to pass through the XBAR.

Terminated Accesses

A master access will be terminated if the **hsel** input of the XBAR is asserted and the transfer type is IDLE. The XBAR will terminated the access and it will not be allowed to pass through the XBAR.

Taken Accesses

A master access will be taken if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case the XBAR will be completely transparent and the master's access will be immediately seen on the slave bus and no arbitration delays will be incurred.

Stalled Accesses

A master access will be stalled if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the access decodes to a slave port that is busy serving another master, parked on another master or is in low power park mode. The XBAR will indicate to the master that the address phase of the access has been taken but will then queue the access to the appropriate slave port to enter into arbitration for access to that slave port.

If the slave port is currently parked on another master or is in low power park mode and no other master is requesting access to the slave port then only one clock of arbitration will be incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters then the master will gain control over the slave port as soon as the data phase of the current access is completed (burst and locked transfers excluded). If the slave port is currently servicing another master of a higher priority then the master will gain control of the slave port once the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

Error Response Terminated Accesses

A master access will be responded to with an error if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the access decodes to a location not occupied by a slave port. This is the only time the XBAR will respond with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

14.6.3 Slave Ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. In order to do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR will force a bubble onto the slave bus when a master is actively making a request. This occurs when a higher priority master has control of the slave port and is running single clock (zero wait state) accesses while a lower priority master is stalled waiting for control of the slave port. When the higher priority master either

leaves the slave port or runs an IDLE cycle to the slave port the XBAR will take control of the slave bus and run a single IDLE cycle before giving the slave port to the lower priority master that was waiting for control of the slave port.

The only other times the XBAR will have control of the slave port is when the XBAR is halting or when no masters are making access requests to the slave port and the XBAR is forced to either park the slave port on a specific master or put the slave port into low power park mode.

In most instances when the XBAR has control of the slave port it will indicate IDLE for the transfer type, negate all control signals and indicate ownership of the slave bus via the **hmaster** encoding of 4'b0000. One exception to this rule is when a master running locked cycles has left the slave port but continues to run locked cycles. In this case the XBAR will control the slave port and will indicate IDLE for the transfer type but it will not affect any other signals.

Note: When a master runs a locked cycle through the XBAR, the master will be guaranteed ownership of all slave ports it accesses while running locked cycles for one cycle beyond when the master finishes running locked cycles.

15 Error Correction Status Module (ECSM)

15.1 Introduction

The Error Correction Status Module (ECSM) provides control functions for the device Standard Product Platform (SPP) including program-visible information about the platform configuration and revision levels, a reset status register, a software watchdog timer, and wakeup control for exiting sleep modes, and optional features such as an address map for the device's crossbar switch, information on memory errors reported by error-correcting codes and/or generic access error information for certain processor cores.

15.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of control functions for the platform device.

15.3 Features

The ECSM includes these features:

- Program-visible information on the platform device configuration and revision
- Reset status register (MRSR)
- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes if error-correcting codes are implemented
- Access address information for faulted memory accesses for certain processor core micro-architectures
- XBAR priority functions, including forcing round robin and high priority enabling

15.4 Memory map and registers description

This section details the programming model for the ECSM. This is an on-platform 128-byte space mapped to the region serviced by an IPS bus controller. Some of the control registers have a 64-bit width. These 64-bit registers are implemented as two 32-bit registers, and include an "H" and "L" suffixes, indicating the "high" and "low" portions of the control function.

The Error Correction Status Module does not include any logic that provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

ECSM registers are accessible only when the core is in supervisor mode.

15.4.1 Memory map

Table 139 lists the registers in the ECSM.

Table 139. ECSM registers

Offset from ECSM_BASE 0xFFF4_0000 (ECSM_0) 0x8FF4_0000 (ECSM_1)	Register	Location	Size (bits)
0x0000	PCT—Processor Core Type register	*on page 341*	16
0x0002	REV—Revision register	*on page 341*	16
0x0004	PLAMC — Platform XBAR Master Configuration	*on page 342*	16
0x0006	PLASC — Platform XBAR Sleave Configuration	*on page 342*	16
0x0008	IMC—IPS Module Configuration register	*on page 343*	16
0x000C–0x000E	Reserved		
0x000F	MRSR—Miscellaneous Reset Status register	*on page 343*	8
0x0010–0x001E	Reserved		
0x001F	MIR—Miscellaneous Interrupt register	*on page 344*	8
0x0020–0x0023	Reserved		
0x0024	MUDCR—Miscellaneous User-Defined Control Register	*on page 345*	32
0x0028–0x0042	Reserved		
0x0043	ECR—ECC Configuration register	*on page 346*	8
0x0044–0x0046	Reserved		
0x0047	ESR—ECC Status register	*on page 347*	8
0x0048–0x0049	Reserved		
0x004A	EEGR—ECC Error Generation register	*on page 349*	16
0x004C–0x004F	Reserved		
0x0050	FEAR—Flash ECC Address register	*on page 351*	32
0x0054–0x0055	Reserved		
0x0056	FEMR—Flash ECC Master Number Register	*on page 352*	8
0x0057	FEAT—Flash ECC Attributes register	*on page 352*	8
0x0058–0x005B	Reserved		
0x005C	FEDR—Flash ECC Data register	*on page 353*	32
0x0060	REAR—RAM ECC Address register	*on page 354*	32
0x0064	Reserved		
0x0065	RESR—RAM ECC Syndrome register	*on page 355*	8
0x0066	REMR—RAM ECC Master register	*on page 357*	8
0x0067	REAT—RAM ECC Attributes register	*on page 357*	8
0x0068–0x006B	Reserved		
0x006C	REDR—RAM ECC Data register	*on page 358*	32
0x0070–0x3FFF	Reserved		

15.4.2 Registers description

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, *writes to the programming model must match the size of the register*, e.g., an *n*-bit register only supports *n*-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

Processor core type (PCT) register

The PCT is a 16-bit read-only register that specifies the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Figure 146. Processor core type (PCT) register

Address: Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PCT[15:0]															
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0

Table 140. PCT field descriptions

Name	Description
0-15 PCT[15:0]	Processor Core Type 0xE012 identifies the z0H Power Architecture.

Revision (REV) register

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Figure 147. Revision (REV) register

Address: Base + 0x0002 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REV[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 141. REV field descriptions

Name	Description
0-15 REV[15:0]	Revision The REV[15:0] field is specified by an input signal to define a software-visible revision number.

Platform XBAR Master Configuration (PLAMC)

The PLAMC is a 16-bit read-only register identifying the presence/absence of bus master connections to the device's AMBA-AHB Crossbar Switch (XBAR). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Figure 148. Platform XBAR Master Configuration (PLAMC) register

Address Base + 0x0004 Access: User read/-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	AMC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	AMC[7:0]							

Table 142. PLAMC field descriptions

Field	Description
AMC[7:0]	XBAR Master Configuration 0 Bus master connection to XBAR input port *n* is not present. 1 Bus master connection to XBAR input port *n* is present.

Platform XBAR Slave Configuration (PLASC)

The PLASC is a 16-bit read-only register identifying the presence/absence of bus slave connections to the device's AMBA-AHB Crossbar Switch (XBAR), plus a 1-bit flag defining the internal platform datapath width (DP64). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Figure 149. Platform XBAR Slave Configuration (PLASC) register

Address Base + 0x0006 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DP64	0	0	0	0	0	0	0	ASC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	ASC[7:0]							

Table 143. ASC field descriptions

Field	Description
DP64	64-bit Datapath 0 Datapath width is 32 bits. 1 Datapath width is 64 bits.
ASC[7:0]	XBAR Slave Configuration 0 Bus slave connection to XBAR output port *n* is not present. 1 Bus slave connection to XBAR output port *n* is present.

IPS Module Configuration (IMC) register

The IMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary slave bus controller. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Figure 150. IPS Module Configuration (IMC) register

Address Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MC[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MC[15:0]															
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 144. IMC field descriptions

Field	Description
0-31 MC[31:0]	IPS Module Configuration 0 IPS module connection to decoded slot *n* not present 1 IPS module connection to decoded slot *n* present

Miscellaneous Reset Status Register (MRSR)

The MRSR contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the MRSR, reflecting the cause of the most recent reset as signaled by device reset input signals. The MRSR can only be read from the IPS programming model. Any attempted write is ignored.

Figure 151. Miscellaneous Reset Status Register (MRSR)

Address: Base + 0x000F Access: User read-only

	0	1	2	3	4	5	6	7
R	POR	DIR	0	0	0	0	0	0
W								
Reset	x	x	0	0	0	0	0	0

Table 145. MRSR field descriptions

Field	Description
0 POR	Power-On Reset 0 Last recorded event was not caused by a power-on reset (based on a device input signal). 1 Last recorded event was caused by a power-on reset (based on a device input signal).
1 DIR	Device Input Reset 0 Last recorded event was not caused by a device input reset. 1 Last recorded event was a reset caused by a device input reset.

Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the MIR register. This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the ECSMIR must be explicitly cleared.

Figure 152. Miscellaneous Interrupt Register (MIR)

Address: Base + 0x001F Access: User read/write

	0	1	2	3	4	5	6	7
R	FB0AI	FB0SI	FB1AI	FB1SI	0	0	0	0
W	1	1	1	1	x	x	x	x
Reset	0	0	0	0	0	0	0	0

Table 146. MIR field descriptions

Field	Description
0 FB0AI	Flash Bank 0 Abort Interrupt 0 A flash bank 0 abort has not occurred. 1 A flash bank 0 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
1 FB0SI	Flash Bank 0 Stall Interrupt 0 A flash bank 0 stall has not occurred. 1 A flash bank 0 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
2 FB1AI	Flash Bank 1 Abort Interrupt 0 A flash bank 1 abort has not occurred. 1 A flash bank 1 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
3 FB1SI	Flash Bank 1 Stall Interrupt 0 A flash bank 1 stall has not occurred. 1 A flash bank 1 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.

Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It provides configuration control for assorted modules on the device. The contents of this register is output from the ECSM to other modules where these user-defined control functions are implemented.

Figure 153. Miscellaneous User-Defined Control register (MUDCR)

Address Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MUDCR[30]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 147. MUDCR field descriptions

Name	Description
1 MUDCR[30]	Pram Wait State Control (MUDCR[30]) When the programmable wait-state pram controller is included on the platform, this bit is used to select whether the pram controller will insert 1-wait state into every read access made to the ram arrays. By setting the hardware define 1 = spp_pram_ecc32 controller is a 1-wait state controller 0 = spp_pram_ecc32 controller is a 0-wait state controller

ECC registers

There are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include the following:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Flash ECC Address Register (FEAR)
- Flash ECC Master Number Register (FEMR)
- Flash ECC Attributes Register (FEAT)
- Flash ECC Data Register (FEDR)
- RAM ECC Address Register (REAR)
- RAM ECC Syndrome Register (RESR)
- RAM ECC Master Number Register (REMR)
- RAM ECC Attributes Register (REAT)
- RAM ECC Data Register (REDR)

The details on the ECC registers are provided in the subsequent sections. If the design does not include ECC on the memories, these addresses are reserved locations within the ECSM's programming model.

ECC Configuration Register (ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches, which are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) that may be useful for subsequent failure analysis.

The reporting of single-bit memory corrections can only be enabled via an SoC-configurable module input signal. While not directly accessible to a user, this capability is viewed as important for error logging and failure analysis.

Figure 154. ECC Configuration register (ECR)

Address: Base + 0x0043 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Table 148. ECR field descriptions

Field	Description
2 ER1BR	Enable RAM 1-bit Reporting This bit can only be set if the input enable signal is asserted. The occurrence of a single-bit RAM correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. 0 Reporting of single-bit RAM corrections disabled 1 Reporting of single-bit RAM corrections enabled
3 EF1BR	Enable Flash 1-bit Reporting This bit can only be set if the input enable signal is asserted. The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0Reporting of single-bit flash corrections disabled 1Reporting of single-bit flash corrections enabled

Table 148. ECR field descriptions (continued)

Field	Description
6 ERNCR	Enable RAM Non-Correctable Reporting The occurrence of a non-correctable multi-bit RAM error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. 0Reporting of non-correctable RAM errors disabled 1Reporting of non-correctable RAM errors enabled
7 EFNCR	Enable Flash Non-Correctable Reporting The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0 Reporting of non-correctable flash errors disabled 1 Reporting of non-correctable flash errors enabled

ECC Status Register (ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly enabled ECC events have been detected. The ESR signals the last properly enabled memory event to be detected. ECC interrupt generation is separated into single-bit error detection/correction, uncorrectable error detection, and the combination of the two as defined by the following boolean equations:

```
ECSM_ECC1BIT_IRQ
    =  ECR[ER1BR] &  ESR[R1BC]// ram, 1-bit correction
    |  ECR[EF1BR] &  ESR[F1BC]// flash, 1-bit correction
ECSM_ECCRNCR_IRQ
    =  ECR[ERNCR] &  ESR[RNCE]// ram, noncorrectable error
ECSM_ECCFNCR_IRQ
    =  ECR[EFNCR] &  ESR[FNCE]// flash, noncorrectable error
ECSM_ECC2BIT_IRQ
    =  ECSM_ECCRNCR_IRQ   // ram, noncorrectable error
    |  ECSM_ECCFNCR_IRQ   // flash, noncorrectable error
ECSM_ECC_IRQ
    =  ECSM_ECC1BIT_IRQ   // 1-bit correction
    |  ECSM_ECC2BIT_IRQ   // noncorrectable error
```

where the combination of a properly enabled category in the ECR and the detection of the corresponding condition in the ESR produces the interrupt request.

The ECSM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly enabled ECC event. If there is a pending ECC interrupt and another properly enabled ECC event occurs, the ECSM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

Figure 155. ECC Status register (ESR)

Address: Base + 0x0047 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W								
Reset	0	0	0	0	0	0	0	0

Table 149. ESR field descriptions

Field	Description
2 R1BC	RAM 1-bit Correction This bit can only be set if ECR[ER1BR] is asserted. The occurrence of a properly enabled single-bit RAM correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit RAM correction detected 1 Reportable single-bit RAM correction detected
3 F1BC	Flash 1-bit Correction This bit can only be set if ECR[EF1BR] is asserted. The occurrence of a properly enabled single-bit flash correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit flash correction detected 1 Reportable single-bit flash correction detected
6 RNCE	RAM Non-Correctable Error The occurrence of a properly enabled non-correctable RAM error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. This bit can only be set if ECR[ERNCR] is asserted. 0 No reportable non-correctable RAM error detected 1 Reportable non-correctable RAM error detected
7 FNCE	Flash Non-Correctable Error The occurrence of a properly enabled non-correctable flash error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. This bit can only be set if ECR[ERNCR] is asserted. 0 No reportable non-correctable flash error detected 1 Reportable non-correctable flash error detected

In the event that multiple status flags are signaled simultaneously, ECSM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

ECC Error Generation Register (EEGR)

The ECC error generation register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the RAM, similar capabilities exist for the flash, that is, the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

The enabling of these error generation modes requires the same input enable signal (as that used to enable single-bit correction reporting) be asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.

Figure 156. ECC Error Generation register (EEGR)

Address Base + 0x004A — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FRC 1BI	FR11 BI	0	0	FRC NCI	FR1 NCI	0	ERRBIT[6:0]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 150. EEGR field descriptions

Field	Description
2 FRC1BI	Force RAM Continuous 1-Bit Data Inversions 0 No RAM continuous 1-bit data inversions generated 1 1-bit data inversions in the RAM continuously generated The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same input enable signal (as that used to enable single-bit correction reporting) is asserted.
3 FR11BI	Force RAM One 1-bit Data Inversion 0 No RAM single 1-bit data inversion generated 1 One 1-bit data inversion in the RAM generated The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same input enable signal (as that used to enable single-bit correction reporting) is asserted.
6 FRCNCI	Force RAM Continuous Non-Correctable Data Inversions 0 No RAM continuous 2-bit data inversions generated 1 2-bit data inversions in the RAM continuously generated The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.

Table 150. EEGR field descriptions (continued)

Field	Description
7 FR1NCI	Force RAM One Non-Correctable Data Inversions 0 No RAM single 2-bit data inversions generated 1 One 2-bit data inversion in the RAM generated The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.
9-15 ERRBIT [6:0]	Error Bit Position The vector defines the bit position that is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted. The RAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 32-bit RAM implementation. The 32-bit ECC approach requires 7 code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32 bits data + 7 bits for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] of the odd bank is inverted. if ERRBIT = 1, then RAM[1] of the odd bank is inverted. ... if ERRBIT = 31, then RAM[31] of the odd bank is inverted. if ERRBIT = 64, then ECC Parity[0] of the odd bank is inverted. if ERRBIT = 65, then ECC Parity[1] of the odd bank is inverted. ... if ERRBIT = 70, then ECC Parity[6] of the odd bank is inverted. For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.

If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion will be generated.

The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in undefined behavior.

Flash ECC Address Register (FEAR)

The FEAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 157. Flash ECC Address register (FEAR)

Address Base + 0x0050 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEAR[31:16]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEAR[15:0]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 151. FEAR field descriptions

Field	Description
0-31 FEAR[31:0]	Flash ECC Address Register This 32-bit register contains the faulting access address of the last properly enabled flash ECC event.

Flash ECC Master Number Register (FEMR)

The FEMR is a 4-bit register for capturing the XBAR bus master number of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 158. Flash ECC Master Number Register (FEMR)

Address: Base + 0x0056 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR[3:0]			
W								
Reset	0	0	0	0	–	–	–	–

Table 152. FEMR field descriptions

Name	Description
4-7 FEMR[3:0]	Flash ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last properly enabled flash ECC event.

Flash ECC Attributes (FEAT) register

The FEAT is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC

Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 159. Flash ECC Attributes (FEAT) Register

Address: Base + 0x0057 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROTECTION[3:0]			
W								
Reset	0	0	0	0	–	–	–	–

Table 153. FEAT field descriptions

Name	Description
0 WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
1-3 SIZE[2:0]	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
4 PROTECTION[3]	AMBA-AHB HPROT[3] Protection[0]: Type 0 I-Fetch 1 Data
5 PROTECTION[2]	AMBA-AHB HPROT[2] Protection[1]: Mode 0 User mode 1 Supervisor mode
6 PROTECTION[1]	AMBA-AHB HPROT[1] Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable
7 PROTECTION[0]	AMBA-AHB HPROT[0] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable

Flash ECC Data Register (FEDR)

The FEDR is a 32-bit register for capturing the data associated with the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 160. Flash ECC Data register (FEDR)

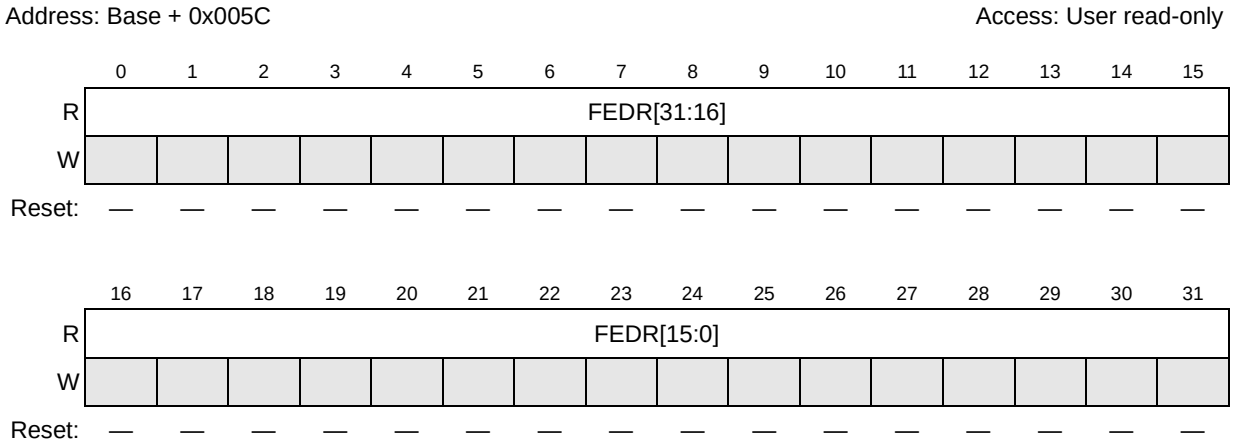


Table 154. FEDR field descriptions

Name	Description
0-31 FEDR[31:0]	Flash ECC Data Register This 32-bit register contains the data associated with the faulting access of the last properly enabled flash ECC event. The register contains the data value taken directly from the data bus.

RAM ECC Address Register (REAR)

The REAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 161. RAM ECC Address register (REAR)

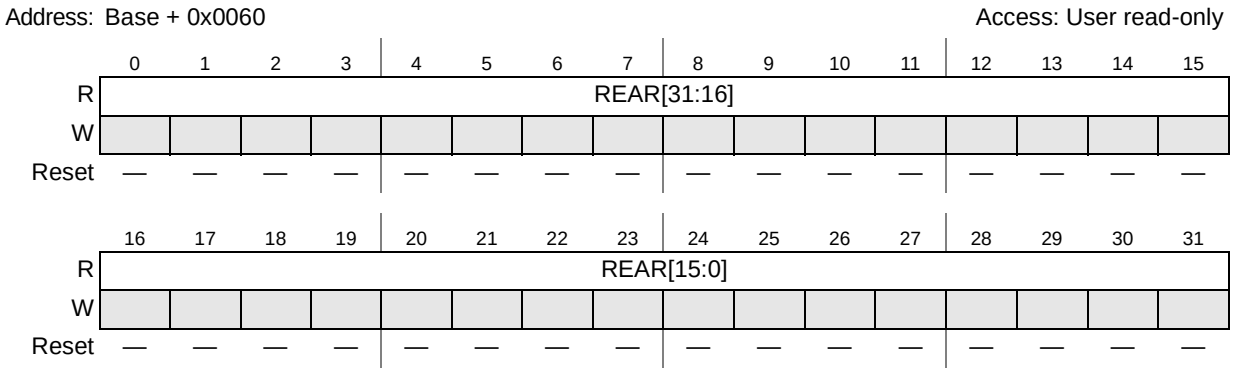


Table 155. REAR field descriptions

Name	Description
0-31 REAR[31:0]	RAM ECC Address Register This 32-bit register contains the faulting access address of the last properly enabled RAM ECC event.

RAM ECC Syndrome Register (RESR)

The RESR is an 8-bit register for capturing the error syndrome of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 162. RAM ECC Syndrome Register (RESR)

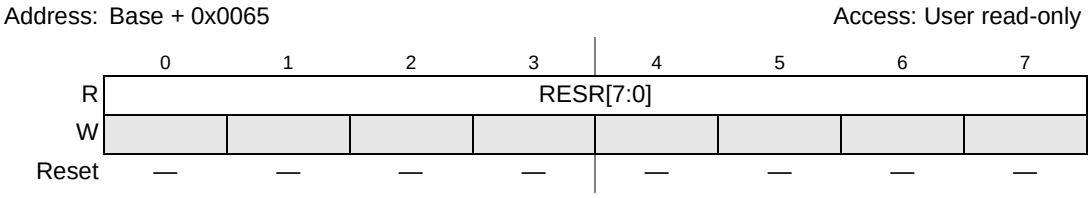


Table 156. RESR field descriptions

Name	Description
0-7 RESR[7:0]	RAM ECC Syndrome Register This 8-bit syndrome field includes 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error. For correctable single-bit errors, the mapping shown in *Table 157* associates the upper 7 bits of the syndrome with the data bit in error.

Note: Table 157 associates the upper 7 bits of the ECC syndrome with the exact data bit in error for single-bit correctable codewords. This table follows the bit vectoring notation where the LSB=0. Note that the syndrome value of 0x0001 implies no error condition but this value is not readable when the PRESR is read for the no error case.

Table 157. RAM syndrome mapping for single-bit correctable errors

RESR[7:0]	Data Bit in Error
0x00	ECC ODD[0]
0x01	No Error
0x02	ECC ODD[1]
0x04	ECC ODD[2]

Table 157. RAM syndrome mapping for single-bit correctable errors (continued)

RESR[7:0]	Data Bit in Error
0x06	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x01	ECC ODD[0]
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x07	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x10	ECC ODD[4]
0x20	ECC ODD[5]
0x40	ECC ODD[6]
0x43	DATA ODD BANK[0]
0x45	DATA ODD BANK[1]
0x46	DATA ODD BANK[2]
0x49	DATA ODD BANK[3]
0x4a	DATA ODD BANK[4]
0x4c	DATA ODD BANK[5]
0x4f	DATA ODD BANK[21]
0x51	DATA ODD BANK[6]
0x52	DATA ODD BANK[7]
0x54	DATA ODD BANK[8]
0x57	DATA ODD BANK[22]
0x58	DATA ODD BANK[9]
0x5b	DATA ODD BANK[23]
0x5d	DATA ODD BANK[24]
0x5e	DATA ODD BANK[25]
0x61	DATA ODD BANK[10]
0x62	DATA ODD BANK[11]
0x64	DATA ODD BANK[12]
0x67	DATA ODD BANK[26]
0x68	DATA ODD BANK[13]
0x6b	DATA ODD BANK[27]
0x6d	DATA ODD BANK[28]
0x6e	DATA ODD BANK[29]
0x70	DATA ODD BANK[14]
0x73	DATA ODD BANK[15]

Table 157. RAM syndrome mapping for single-bit correctable errors (continued)

RESR[7:0]	Data Bit in Error
0x75	DATA ODD BANK[16]
0x76	DATA ODD BANK[17]
0x79	DATA ODD BANK[18]
0x7a	DATA ODD BANK[19]
0x7c	DATA ODD BANK[20]
0x7f	DATA ODD BANK[30]
0x03,0x05.....0x4D	Multiple bit error
> 0x4D	Multiple bit error

RAM ECC Master Number Register (REMR)

The REMR is an 8-bit register in which the 4-bit field REMR[0:3] is used for capturing the XBAR bus master number of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 163. RAM ECC Master Number register (REMR)

Address: Base + 0x0066 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR[3:0]			
W								
Reset	0	0	0	0	—	—	—	—

Table 158. REMR field descriptions

Name	Description
4-7 REMR[3:0]	RAM ECC Master Number Register This 4-bit field contains the XBAR bus master number of the faulting access of the last properly enabled RAM ECC event.

RAM ECC Attributes (REAT) register

The REAT is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 164. RAM ECC Attributes (REAT) register

Address: Base + 0x0067 Access: User read/write

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROTECTION[3:0]			
W								
Reset	—	—	—	—	—	—	—	—

Table 159. REAT field descriptions

Name	Description
0 WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
1-3 SIZE[2:0]	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
4 PROTECTION[3]	AMBA-AHB HPROT[3] Protection[0]: Type 0 I-Fetch 1 Data
5 PROTECTION[2]	AMBA-AHB HPROT[2] Protection[1]: Mode 0 User mode 1 Supervisor mode
6 PROTECTION[1]	AMBA-AHB HPROT[1] Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable
7 PROTECTION[0]	AMBA-AHB HPROT[0] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable

RAM ECC Data Register (REDR)

The REDR is a -bit register for capturing the data associated with the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 165. Platform RAM ECC Data register (PREDR)

Address: Base + 0x006C Access: User read-only

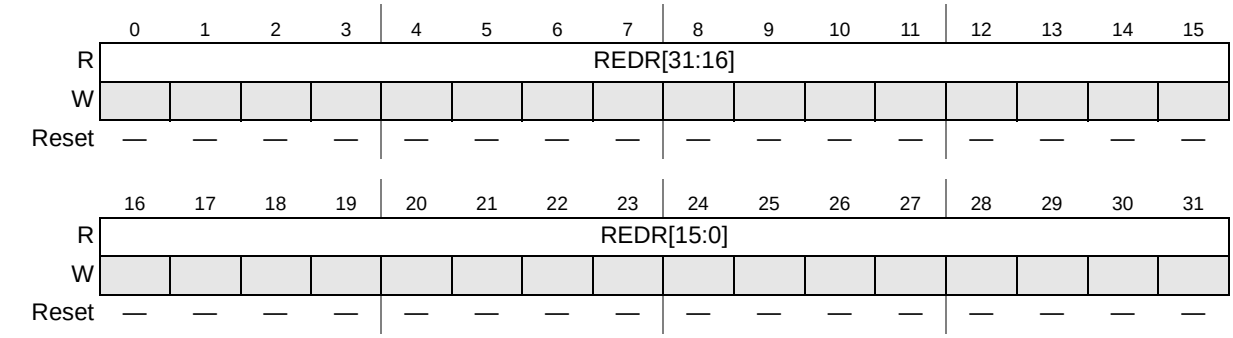


Table 160. REDR field descriptions

Name	Description
0-31 REDR[31:0]	RAM ECC Data Register This 32-bit register contains the data associated with the faulting access of the last properly enabled RAM ECC event. The register contains the data value taken directly from the data bus.

16 Internal Static RAM (SRAM)

16.1 Introduction

The general-purpose SRAM has a size of 80 KB. The first SRAM block (48 KB) is connected to SRAMC_0 (SRAM controller 0) and the second SRAM block (32 KB) is connected to SRAMC_1 (SRAM controller 1)

The SRAM provides the following features:

- SRAM can be read/written from any bus master
- Byte, halfword, word and doubleword addressable
- Single-bit correction and double-bit error detection

16.2 SRAM operating mode

The SRAM has only one operating mode. No standby mode is available.

Table 161. SRAM operating modes

Mode	Configuration
Normal (functional)	Allows reads and writes of SRAM

16.3 Module memory map

The SRAM occupies up to 80 KB of address space.

Table 162 shows the SRAM memory map.

Table 162. SRAM memory map

Address	Use
0x4000_0000 (Base)	48 KB RAM
0x5000_0000 (Base)	32 KB RAM

16.4 Register descriptions

The SRAM has no registers. Registers associated with the ECC are located in the ECSM. See *Section : ECC registers*.

16.5 SRAM ECC mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 39-bit reads (32-bit data bus plus the 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than 2 bits.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 32-bit data width (1 or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1 or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

16.5.1 Access timing

The system bus is a two-stage pipelined bus that makes the timing of any access dependent on the access during the previous clock. *Table 163* lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

- Current operation—Lists the type of SRAM operation currently executing
- Previous operation—Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)
- Wait states—Lists the number of wait states (bus clocks) the operation requires, which depends on the combination of the current and previous operation

Table 163. Number of wait states required for SRAM operations

Operation type	Current operation	Previous operation	Number of wait states required
Read	Read	Idle	1
		Pipelined read	
		8 , 16 or 32-bit write	0 (read from the same address)
			1 (read from a different address)
	Pipelined read	Read	0

Table 163. Number of wait states required for SRAM operations (continued)

Operation type	Current operation	Previous operation	Number of wait states required
Write	8 or 16-bit write	Idle	1
		Read	
		Pipelined 8- or 16-bit write	2
		32-bit write	
		8 or 16-bit write	0 (write to the same address)
	Pipelined 8, 16 or 32-bit write	8 , 16 or 32-bit write	0
	32-bit write	Idle	0
		32-bit write	
		Read	

16.5.2 Reset effects on SRAM accesses

Asynchronous reset will possibly corrupt RAM if it asserts during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. If no access is occurring when reset occurs, RAM corruption does not happen.

Instead synchronous reset (SW reset) should be used in controlled function (without RAM accesses) in case initialization procedure is needed without RAM initialization.

16.6 Functional description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a R/W operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 32-bit write operations prior any read accesses. This is also true for implicit read accesses caused by any write accesses smaller than 32 bits as discussed in *Section 16.5: SRAM ECC mechanism*.

16.7 Initialization and application information

To use the SRAM, the ECC must check all bits that require initialization after power on. All writes must specify an even number of registers performed on 32-bit word-aligned boundaries. If the write is not the entire 32-bits (8 or 16 bits), a read/modify/write operation is generated that checks the ECC value upon the read. Refer to *Section 16.5: SRAM ECC mechanism*.

Note: You must initialize SRAM, even if the application does not use ECC reporting.

17 Flash Memory

17.1 Introduction

The Flash memory comprises a platform Flash controller interface and two Flash memory arrays: two arrays of 512 KB for code (code Flash) and one array of 64 KB for data (data Flash). The Flash architecture of the SPC56xP60/54 device is illustrated in *Figure 166*.

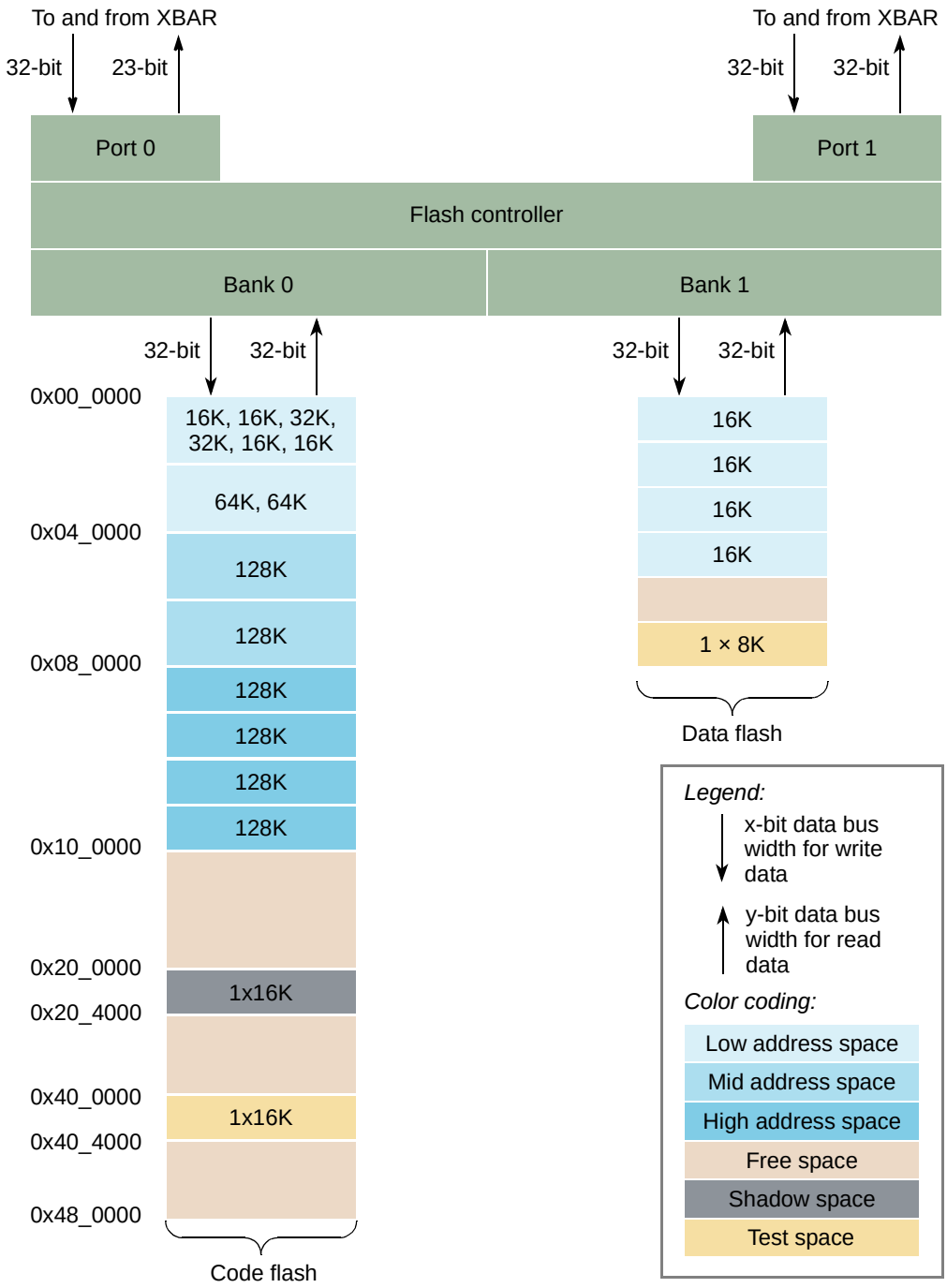


Figure 166. SPC56xP60/54 Flash memory architecture

17.2 Platform Flash controller

17.2.1 Introduction

This section provides an introduction of the platform Flash controller, which acts as the interface between the system bus and as many as three banks of Flash memory arrays (program and data). It intelligently converts the protocols between the system bus and the dedicated Flash array interfaces. Several important terms are used to describe the platform Flash controller module and its connections. These terms are defined here.

- **Port**—This term describes the AMBA-AHB connection(s) into the platform Flash controller. From an architectural and programming model viewpoint, the definition supports as many as two AHB ports, even though this specific controller only supports a dual AHB connections.
- **Bank**—This term describes the attached Flash memories. From the platform Flash controller's perspective, there may be one or two attached banks of Flash memory. The code Flash bank is required and always attached to bank0. Additionally, there is a data Flash attached to bank1. The platform Flash controller interface supports two separate connections, one to each memory bank. On the SPC56xP60/54 device, bank0 and bank1 are internal to the device.
- **Array**—Each memory bank has one Flash array instantiation.
- **Page**—This value defines the number of bits read from the Flash array in a single access. For this controller and memory, the page size is 128 bits (16 bytes).

The nomenclature "page buffers" and "line buffers" are used interchangeably.

Overview

The platform Flash controller supports a 32-bit data bus width at the AHB port and connections to 128-bit read data interfaces from two memory banks, where each bank contains one instantiation of the Flash memory array. One Flash bank is connected to the code Flash memory and the other bank is connected to the data Flash memory. The memory controller capabilities vary between the two banks with each bank's functionality optimized with the typical use cases associated with the attached Flash memory. As an example, the platform Flash controller logic associated with the code Flash bank contains a four-entry "page" buffer, each entry containing 128 bits of data (1 Flash page) plus an associated controller that prefetches sequential lines of data from the Flash array into the buffer, while the controller logic associated with the data Flash bank only supports a 128-bit register that serves as a temporary page holding register and does not support any prefetching. Prefetch buffer hits from the code Flash bank support 0-wait AHB data phase responses. AHB read requests that miss the buffers generate the needed Flash array access and are forwarded to the AHB upon completion, typically incurring two wait states at an operating frequency of 60 to 64 MHz.

This memory controller is optimized for applications where a cacheless processor core, for example the Power e200z0h, is connected through the platform to on-chip memories, for example Flash and RAM, where the processor and platform operate at the same frequency. For these applications, the 2-stage pipeline AMBA-AHB system bus is effectively mapped directly into stages of the processor's pipeline and 0 wait state responses for most memory accesses are critical for providing the required level of system performance.

Features

The following list summarizes the key features of the platform Flash controller:

- Single AHB port interface supports a 32-bit data bus. All AHB aligned and unaligned reads within the 32-bit container are supported. Only aligned word writes are supported.
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank.
- Interface with code Flash (bank0) provides configurable read buffering and page prefetch support. Four page read buffers (each 128 bits wide) and a prefetch controller support single-cycle read responses (0 AHB data phase wait states) for hits in the buffers. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- Interface with data Flash (bank1) includes a 128-bit register to temporarily hold a single Flash page. This logic supports single-cycle read responses (0 AHB data phase wait states) for accesses that hit in the holding register. There is no support for prefetching associated with bank1.
- Programmable response for read-while-write sequences including support for stall-while-write, optional stall notification interrupt, optional Flash operation termination, and optional termination notification interrupt
- Separate and independent configurable access timing (on a per bank basis) to support use across a wide range of platforms and frequencies
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit Flash ECC events
- Typical operating configuration loaded into programming model by system reset

17.2.2 Modes of operation

The platform Flash controller module does not support any special modes of operation. Its operation is driven from the AMBA-AHB memory references it receives from the platform's bus masters. Its configuration is defined by the setting of the programming model registers, physically located as part of the Flash array modules.

17.2.3 External signal descriptions

The platform Flash controller does not directly interface with any external signals. Its primary internal interfaces include a connection to an AMBA-AHB crossbar (or memory protection unit) slave port and connections with as many as two banks (code and data) of Flash memory, each containing one instantiation of the Flash array. Additionally, the operating configuration for the platform Flash controller is defined by the contents of certain code Flash array0 registers that are inputs to the module.

17.2.4 Memory map and registers description

Two memory maps are associated with the platform Flash controller: one for the Flash memory space and another for the program-visible control and configuration registers. The Flash memory space is accessed via the AMBA-AHB port. The program-visible registers are accessed via the slave peripheral bus. Details on both memory spaces are provided in *Section : Memory map.*

There are no program-visible registers that physically reside inside the platform Flash controller. Rather, the platform Flash controller receives control and configuration information from the Flash array controller(s) to determine the operating configuration. These are part of the Flash array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

Note: Updating the configuration fields that control the platform flash controller behavior should only occur while the flash controller is idle. Changing configuration settings while a flash access is in progress can lead to non-deterministic behavior.

Memory map

First, consider the Flash memory space accessed via transactions from the platform Flash controller's AHB port. To support the two separate Flash memory banks, the platform Flash controller uses address bit 23 (haddr[23]) to steer the access to the appropriate memory bank. In addition to the actual Flash memory regions, there are shadow and test sectors included in the system memory map. The program-visible control and configuration registers associated with each memory array are included in the slave peripheral address region. The system memory map defines one code Flash array and one data Flash array. See *Table 164*.

Caution:

Software executing from flash memory must not write to registers that control flash behavior (such as wait state settings or prefetch enable/disable). Doing so can cause data corruption. On this chip, these registers include PFCR0 and PFAPR.

Note: Flash memory configuration registers should be written only with 32-bit write operations to avoid any issues associated with register incoherency caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

Table 164. Flash-related regions in the system memory map

Start address	End address	Size (KB)	Region
0x0000_0000	0x0007_FFFF	512	Code Flash array 0
0x0008_0000	0x000F_FFFF	512	Code Flash array 1
0x0010_0000	0x001F_FFFF	1024	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash array 0: shadow sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash array 0: test sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_FFFF	64	Data Flash array 0
0x0081_0000	0x00C0_1FFF	4040	Reserved
0x00C0_2000	0x00C0_3FFF	8	Data Flash array 0: test sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Emulation Mapping
0xFFE8_8000	0xFFE8_BFFF	16	Code Flash array 0 configuration(1)

Table 164. Flash-related regions in the system memory map (continued)

Start address	End address	Size (KB)	Region
0xFFE8_C000	0xFFE8_FFFF	16	Data Flash array 0 configuration(1)
0xFFEB_0000	0xFFEB_BFFF	48	Reserved

1. This region is also aliased to address 0xC3F8_nnnn.

For additional information on the address-based read access timing for emulation of other memory types, see *Section 17.2.17: Wait state emulation.*

Next, consider the memory map associated with the control and configuration registers.

There are multiple registers that control operation of the platform Flash controller. Note the first two Flash array registers (PFCR0, PFCR1) are reset to a device-defined value, while the remaining register (PFAPR) is loaded at reset from specific locations in the array's shadow region.

Regardless of the number of populated banks or the number of Flash arrays included in a given bank, the configuration of the platform Flash controller is wholly specified by the platform Flash controller control registers associated with code Flash array0. The code array0 register settings define the operating behavior of **both** Flash banks. It is recommended to set the platform Flash controller control registers for both arrays to the array0 values.

Note: To perform program and erase operations, the control registers in the actual referenced Flash array must both be programmed, but the configuration of the platform Flash controller module is defined by the platform Flash controller control registers of code array0.

The 32-bit memory map for the platform Flash controller control registers is shown in *Table 165*.

Table 165. Platform Flash controller 32-bit memory map

Offset from PFlash_BASE (0xFFE8_8000)	Register	Location
0x001C	Platform Flash Configuration Register 0 (PFCR0)	*on page 411*
0x0020	Platform Flash Configuration Register 1 (PFCR1)	*on page 416*
0x0024	Platform Flash Access Protection Register (PFAPR)	*on page 418*

17.2.5 Functional description

The platform Flash controller interfaces between the AHB-Lite 2.v6 system bus and the Flash memory arrays.

The platform Flash controller generates read and write enables, the Flash array address, write size, and write data as inputs to the Flash array. The platform Flash controller captures read data from the Flash array interface and drives it onto the AHB. As much as four pages of data (128-bit width) from bank0 are buffered by the platform Flash controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (0 AHB wait states) read data responses on buffer hits.

Several prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch.

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms.

Throughout this discussion, bk*n*_ is used as a prefix to refer to two signals, each for each bank: bk0_ and bk1_. Also, the nomenclature B*x*_P*y*_RegName is used to reference a program-visible register field associated with bank "*x*" and port "*y*".

17.2.6 Basic interface protocol

The platform Flash controller interfaces to the Flash array by driving addresses (bk*n*_fl_addr[23:0]) and read or write enable signals (bk*n*_fl_rd_en, bk*n*_fl_wr_en).

The read or write enable signal (bk*n*_fl_rd_en, bk*n*_fl_wr_en) is asserted in conjunction with the reference address for a single rising clock when a new access request is made.

Addresses are driven to the Flash array in a flow-through fashion to minimize array access time. When no outstanding access is in progress, the platform Flash controller drives addresses and asserts bk*n*_fl_rd_en or bk*n*_fl_wr_en and then may change to the next outstanding address in the next cycle.

Accesses are terminated under control of the appropriate read/write wait state control setting. Thus, the access time of the operation is determined by the settings of the wait state control fields. Access timing can be varied to account for the operating conditions of the device (frequency, voltage, temperature) by appropriately setting the fields in the programming model for either bank.

The platform Flash controller also has the capability of extending the normal AHB access time by inserting additional wait states for reads and writes. This capability is provided to allow emulation of other memories that have different access time characteristics. The added wait state specifications are provided by bit 28 to bit 24 of Flash address (haddr[28:24], see *Table 167* and *Table 168*). These wait states are applied in addition to the normal wait states incurred for Flash accesses. Refer to *Section 17.2.17: Wait state emulation* for more details.

Prefetching of next sequential page is blocked when haddr[28:24] is non-zero. Buffer hits are also blocked as well, regardless of whether the access corresponds to valid data in one of the page read buffers. These steps are taken to ensure that timing emulation is correct and that excessive prefetching is avoided. In addition, to prevent erroneous operation in certain rare cases, the buffers are invalidated on any non-sequential AHB access with a non-zero value on haddr[28:24].

17.2.7 Access protections

The platform Flash controller provides programmable configurable access protections for both read and write cycles from masters via the Platform Flash Access Protection Register (PFAPR). It allows restriction of read and write requests on a per-master basis. This functionality is described in *Section : Platform Flash Access Protection Register (PFAPR).* Detection of a protection violation results in an error response from the platform Flash controller on the AHB transfer.

17.2.8 Read cycles — buffer miss

Read cycles from the Flash array are initiated by driving a valid access address on bk*n*_fl_addr[23:0] and asserting bk*n*_fl_rd_en for the required setup (and hold) time before (and after) the rising edge of hclk. The platform Flash controller then waits for the programmed number of read wait states before sampling the read data on bk*n*_fl_rdata[127:0]. This data is normally stored in the least-recently updated page read buffer for bank0 in parallel with the requested data being forwarded to the AHB. For bank1, the data is captured in the page-wide temporary holding register as the requested data is forwarded to the AHB bus. Timing diagrams of basic read accesses from the Flash array are shown in *Figure 167* through *Figure 170*.

If the Flash access was the direct result of an AHB transaction, the page buffer is marked as most-recently-used as it is being loaded. If the Flash access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

17.2.9 Read cycles — buffer hit

Single cycle read responses to the AHB are possible with the platform Flash controller when the requested read access was previously loaded into one of the bank0 page buffers. In these “buffer hit” cases, read data is returned to the AHB data phase with a 0 wait state response.

Likewise, the bank1 logic includes a single 128-bit temporary holding register and sequential accesses that “hit” in this register are also serviced with a 0 wait state response.

17.2.10 Write cycles

In a write cycle, address, write data, and control signals are launched off the same edge of hclk at the completion of the first AHB data phase cycle. Write cycles to the Flash array are initiated by driving a valid access address on bk*n*_fl_addr[23:0], driving write data on bk*n*_fl_wdata[63:0], and asserting bk*n*_fl_wr_en. Again, the controller drives the address and control information for the required setup time before the rising edge of hclk, and provides the required amount of hold time. The platform Flash controller then waits for the appropriate number of write wait states before terminating the write operation. On the cycle following the programmed wait state value, the platform Flash controller asserts hready_out to indicate to the AHB port that the cycle has terminated.

17.2.11 Error termination

The platform Flash controller follows the standard procedure when an AHB bus cycle is terminated with an ERROR response. First, the platform Flash controller asserts hresp[0] and negates hready_out to signal an error has occurred. On the following clock cycle, the platform Flash controller asserts hready_out and holds both hresp[0] and hready_out asserted until hready_in is asserted.

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the platform Flash controller does not initiate a Flash array access.

The second case that can cause an error response to the AHB is when an access is performed to the Flash array and is terminated with a Flash error response. See

Section 17.2.13: Flash error response operation. This may occur for either a read or a write operation.

The third case that can cause an error response to the AHB is when a write access is attempted to the Flash array and is disallowed by the state of the bk*n*_fl_ary_access control input. This case is similar to case 1.

A fourth case involves an attempted read access while the Flash array is busy doing a write (program) or erase operation if the appropriate read-while-write control field is programmed for this response. The 3-bit read-while-write control allows for immediate termination of an attempted read, or various stall-while-write/erase operations are occurring.

The platform Flash controller can also terminate the current AHB access if hready_in is asserted before the end of the current bus access. While this circumstance should not occur, this does not result in an error condition being reported, as this behavior is initiated by the AHB. In this circumstance, the platform Flash controller control state machine completes any Flash array access in progress (without signaling the AHB) before handling a new access request.

17.2.12 Access pipelining

The platform Flash controller does not support access pipelining since this capability is not supported by the Flash array. As a result, the APC (Address Pipelining Control) field should typically be the same value as the RWSC (Read Wait State Control) field for best performance, that is, BK*n*_APC = BK*n*_RWSC. It cannot be less than the RWSC.

17.2.13 Flash error response operation

The Flash array may signal an error response by asserting bk*n*_fl_xfr_err to terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the platform Flash controller does not update or validate a bank0 page read buffer nor the bank1 temporary holding register. An error response may be signaled on read or write operations. For more information on the specifics related to signaling of errors, including Flash ECC, refer to subsequent sections in this chapter. For additional information on the system registers that capture the faulting address, attributes, data and ECC information, see *Chapter 15: Error Correction Status Module (ECSM).*

17.2.14 Bank0 page read buffers and prefetch operation

The logic associated with bank0 of the platform Flash controller contains four 128-bit page read buffers that hold data read from the Flash array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

The organization of each page buffer is described as follows in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags.

```
struct {        // bk0_page_buffer
     reg  addr[23:4];// page address
     reg  valid;     // valid bit
     reg  rdata[127:0];// page read data
     reg  xfr_error; // transfer error indicator from Flash array
     reg  multi_ecc_error;// multi-bit ECC error indicator from Flash array
```

```
     reg  single_ecc_error;// single-bit correctable ECC indicator from
Flash array
}    bk0_page_buffer[4];
```

For the general case, a page buffer is written at the completion of an error-free Flash access and the valid bit asserted. Subsequent Flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in *Section 17.2.13: Flash error response operation* a page buffer is *not* marked as valid if the Flash array access terminated with any type of transfer error. However, the result is that Flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see *Section : Buffer invalidation.*

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters may be enabled or disabled from triggering prefetches, and triggering may be further restricted based on whether a read access is for instruction or data. A read access to the platform Flash controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a Flash array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers may be in one of six states, listed here in prioritized order:

1. Invalid—the buffer contains no valid data.
2. Used—the buffer contains valid data that has been provided to satisfy an AHB burst type read.
3. Valid—the buffer contains valid data that has been provided to satisfy an AHB single type read.
4. Prefetched—the buffer contains valid data that has been prefetched to satisfy a potential future AHB access.
5. Busy AHB—the buffer is currently being used to satisfy an AHB burst read.
6. Busy Fill—the buffer has been allocated to receive data from the Flash array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement.

Once the candidate page buffer has been selected, the Flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently-used status is not changed. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of Flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted Flash access.

Several algorithms are available for prefetch control that trade off performance versus power. They are defined by the B*x*_P*y*_PFLM (prefetch limit) register field. More aggressive prefetching increases power slightly due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, a number of control bits must be enabled. Specifically, the global buffer enable (B*x*_P*y*_BFE) must be set, the prefetch limit (B*x*_P*y*_PFLM) must be non-zero and either instruction prefetching (B*x*_P*y*_IPFE) or data prefetching (B*x*_P*y*_DPFE) enabled. Refer to *Section 17.3.6: Registers description* for a description of these control fields.

Instruction/data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the B*x*_P*y*_IPFE control field, while prefetching for data reads is enabled via the B*x*_P*y*_DPFE control field. Additionally, the B*x*_P*y*_PFLIM field must also be set to enable prefetching. Prefetches are never triggered by write cycles.

Per-master prefetch triggering

Prefetch triggering may be also controlled for individual bus masters. AHB accesses indicate the requesting master via the hmaster[3:0] inputs. Refer to *Section : Platform Flash Access Protection Register (PFAPR)* for details on these controls.

Buffer allocation

Allocation of the line read buffers is controlled via page buffer configuration (B*x*_P*y*_BCFG) field. This field defines the operating organization of the four page buffers. The buffers can be organized as a “pool” of available resources (with all four buffers in the pool) or with a fixed partition between buffers allocated to instruction or data accesses. For the fixed partition, two configurations are supported. In one configuration, buffers 0 and 1 are allocated for instruction fetches and buffers 2 and 3 for data accesses. In the second configuration, buffers 0, 1, and 2 are allocated for instruction fetches and buffer 3 reserved for data accesses.

Buffer invalidation

The page read buffers may be invalidated under hardware or software control.

Any falling edge transition of the array’s bk*n*_fl_done signal causes the page read buffers to be marked as invalid. This input is negated by the Flash array at the beginning of all program/erase operations as well as in certain other cases. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer in progress.

Software may invalidate the buffers by clearing the B*x*_P*y*_BFE bit, which also disables the buffers. Software may then re-assert the B*x*_P*y*_BFE bit to its previous state, and the buffers will have been invalidated.

One special case needing software invalidation relates to page buffer “hits” on Flash data that was tagged with a single-bit ECC event on the original array access. Recall that the page buffer structure includes an status bit signaling the array access detected and corrected a single-bit ECC error. On all subsequent buffer hits to this type of page data, a single-bit ECC event is signaled by the platform Flash controller. Depending on the specific hardware configuration, this reporting of a single-bit ECC event may generate an ECC alert

interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

Finally, the buffers are invalidated by hardware on any non-sequential access with a non-zero value on haddr[28:24] to support wait state emulation.

17.2.15 Bank1 temporary holding register

Recall the bank1 logic within the Flash includes a single 128-bit data register, used for capturing read data. Since this bank does not support prefetching, the read data for the referenced address is bypassed directly back to the AHB data bus. The page is also loaded into the temporary data register and subsequent accesses to this page can hit from this register, if it is enabled (B1_Py_BFE).

The organization of the temporary holding register is described as follows, in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags and is the same as an individual bank0 page buffer.

```
struct {        // bk1_page_buffer
     reg  addr[23:4];// page address
     reg  valid;     // valid bit
     reg  rdata[127:0];// page read data
     reg  xfr_error; // transfer error indicator from Flash array
     reg  multi_ecc_error;// multi-bit ECC error indicator from Flash array
     reg  single_ecc_error;// single-bit correctable ECC indicator from
Flash array
}    bk1_page_buffer;
```

For the general case, a temporary holding register is written at the completion of an error-free Flash access and the valid bit asserted. Subsequent Flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait states as the stored read data is routed from the temporary register back to the requesting bus master.

The contents of the holding register are invalidated by the falling edge transition of bk1_fl_done and on any non-sequential access with a non-zero value on haddr[28:24] (to support wait state emulation) in the same manner as the bank0 page buffers. Additionally, the B1_Py_BFE register bit can be cleared by software to invalidate the contents of the holding register.

As noted in *Section 17.2.13: Flash error response operation* the temporary holding register is *not* marked as valid if the Flash array access terminated with any type of transfer error. However, the result is that Flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the temporary holding register and validated. Accordingly, one special case needing software invalidation relates to holding register “hits” on Flash data that was tagged with a single-bit ECC event. Depending on the specific hardware configuration, the reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

The bank1 temporary holding register effectively operates like a single page buffer.

17.2.16 Read-While-Write functionality

The platform Flash controller supports various programmable responses for read accesses while the Flash is busy performing a write (program) or erase operation. For all situations,

the platform Flash controller uses the state of the Flash array's bk*n*_fl_done output to determine if it is busy performing some type of high-voltage operation, namely, if bk*n*_fl_done = 0, the array is busy.

Specifically, there are two 3-bit read-while-write (BK*n*_RWWC) control register fields that define the platform Flash controller's response to these types of access sequences. There are five unique responses that are defined by the BK*n*_RWWC setting: one immediately reports an error on an attempted read, and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- BK*n*_RWWC = 0b0*xx*
  - For this mode, any attempted Flash read to a busy array is immediately terminated with an AHB error response and the read is blocked in the controller and not seen by the Flash array.
- BK*n*_RWWC = 0b111
  - This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform Flash controller module stalls any read reference until the Flash has completed its program/erase operation. If a read access arrives while the array is busy or if a falling-edge on bk*n*_fl_done occurs while a read is still in progress, the AHB data phase is stalled by negating hready_out and saving the address and attributes into holding registers. Once the array has completed its program/erase operation, the platform Flash controller uses the saved address and attribute information to create a pseudo address phase cycle to "retry" the read reference and sends the registered information to the array as bk*n*_fl_rd_en is asserted. Once the retried address phase is complete, the read is processed normally and once the data is valid, it is forwarded to the AHB bus and hready_out negated to terminate the system bus transfer.
- BK*n*_RWWC = 0b110
  - This setting is similar to the basic stall-while-write capability provided when BK*n*_RWWC = 0b111 with the added ability to generate a notification interrupt if a read arrives while the array is busy with a program/erase operation. There are two notification interrupts, one for each bank.
- BK*n*_RWWC = 0b101
  - Again, this setting provides the basic stall-while-write capability with the added ability to terminate any program/erase operation if a read access is initiated. For this setting, the read request is captured and retried as described for the basic stall-while-write, plus the program/erase operation is terminated by the platform Flash controller's assertion of the bkc_fl_abort signal. The bk*n*_fl_abort signal remains asserted until bk*n*_fl_done is driven high. For this setting, there are no notification interrupts generated.
- BK*n*_RWWC = 0b100
  - This setting provides the basic stall-while-write capability with the ability to terminate any program/erase operation if a read access is initiated plus the generation of a termination notification interrupt. For this setting, the read request is captured and retried as described for the basic stall-while-write, the program/erase operation is terminated by the platform Flash controller's assertion of the bk*n*_fl_abort signal and a termination notification interrupt generated. There are two termination notification interrupts, one for each bank.

As detailed above, there are a total of four interrupt requests associated with the stall-while-write functionality. These interrupt requests are captured as part of ECSM's Interrupt Register and logically summed together to form a single request to the interrupt controller.

Table 166. Platform Flash controller stall-while-write interrupts

MIR[*n*]	Interrupt description
ECSM.MIR[7]	Platform Flash bank0 termination notification, MIR[FB0AI]
ECSM.MIR[6]	Platform Flash bank0 stall notification, MIR[FB0SI]
ECSM.MIR[5]	Platform Flash bank1 termination notification, MIR[FB1AI]
ECSM.MIR[4]	Platform Flash bank1 stall notification, MIR[FB1S1]

For example timing diagrams of the stall-while-write and terminate-while-write operations, see *Figure 171* and *Figure 172* respectively.

17.2.17 Wait state emulation

Emulation of other memory array timings are supported by the platform Flash controller on read cycles to the Flash. This functionality may be useful to maintain the access timing for blocks of memory that were used to overlay Flash blocks for the purpose of system calibration or tuning during code development.

The platform Flash controller inserts additional wait states according to the values of haddr[28:24],where haddr represents the Flash address. When these inputs are non-zero, additional cycles are added to AHB read cycles. Write cycles are not affected. In addition, no page read buffer prefetches are initiated, and buffer hits are ignored.

Table 167 and *Table 168* show the relationship of haddr[28:24] to the number of additional primary wait states. These wait states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note that the wait state specification consists of two components: haddr[28:26] and haddr[25:24] and effectively extends the Flash read by (8 × haddr[25:24] + haddr[28:26]) cycles.

Table 167. Additional wait state encoding

Memory address haddr[28:26]	Additional wait states
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 168 shows the relationship of haddr[25:24] to the number of additional wait states. These are applied in addition to those specified by haddr[28:26] and thus extend the total wait state specification capability.

Table 168. Extended additional wait state encoding

Memory address haddr[25:24]	Additional wait states (added to those specified by haddr[28:26])
00	0
01	8
10	16
11	24

17.2.18 Timing diagrams

Since the platform Flash controller is typically used in platform configurations with a cacheless core, the operation of the processor accesses to the platform memories, for example Flash and SRAM, plays a major role in the overall system performance. Given the core/platform pipeline structure, the platform's memory controllers (PFlash, PRAM) are designed to provide a 0 wait state data phase response to maximize processor performance. The following diagrams illustrate operation of various cycle types and responses referenced earlier in this chapter including stall-while-read (*Figure 171*) and terminate-while-read (*Figure 172*) diagrams.

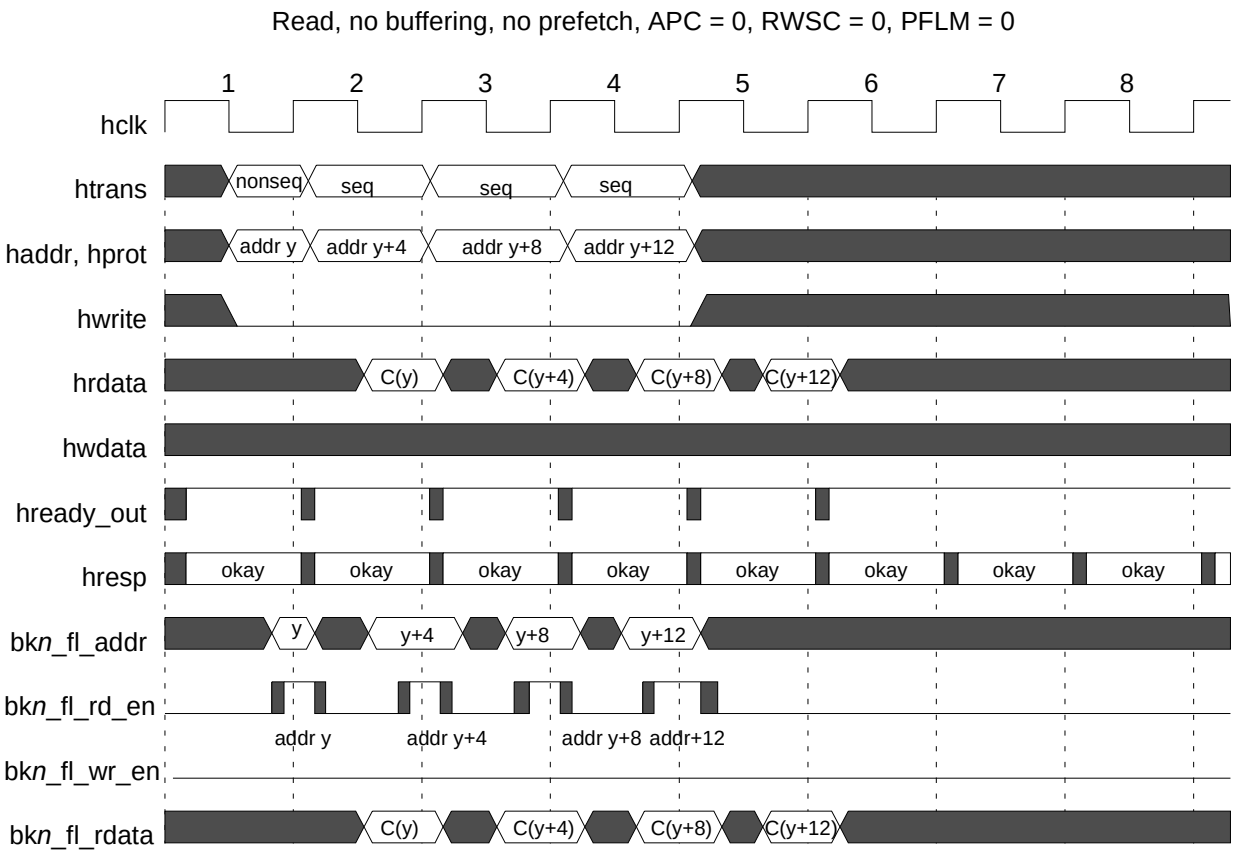


Figure 167. 1-cycle access, no buffering, no prefetch

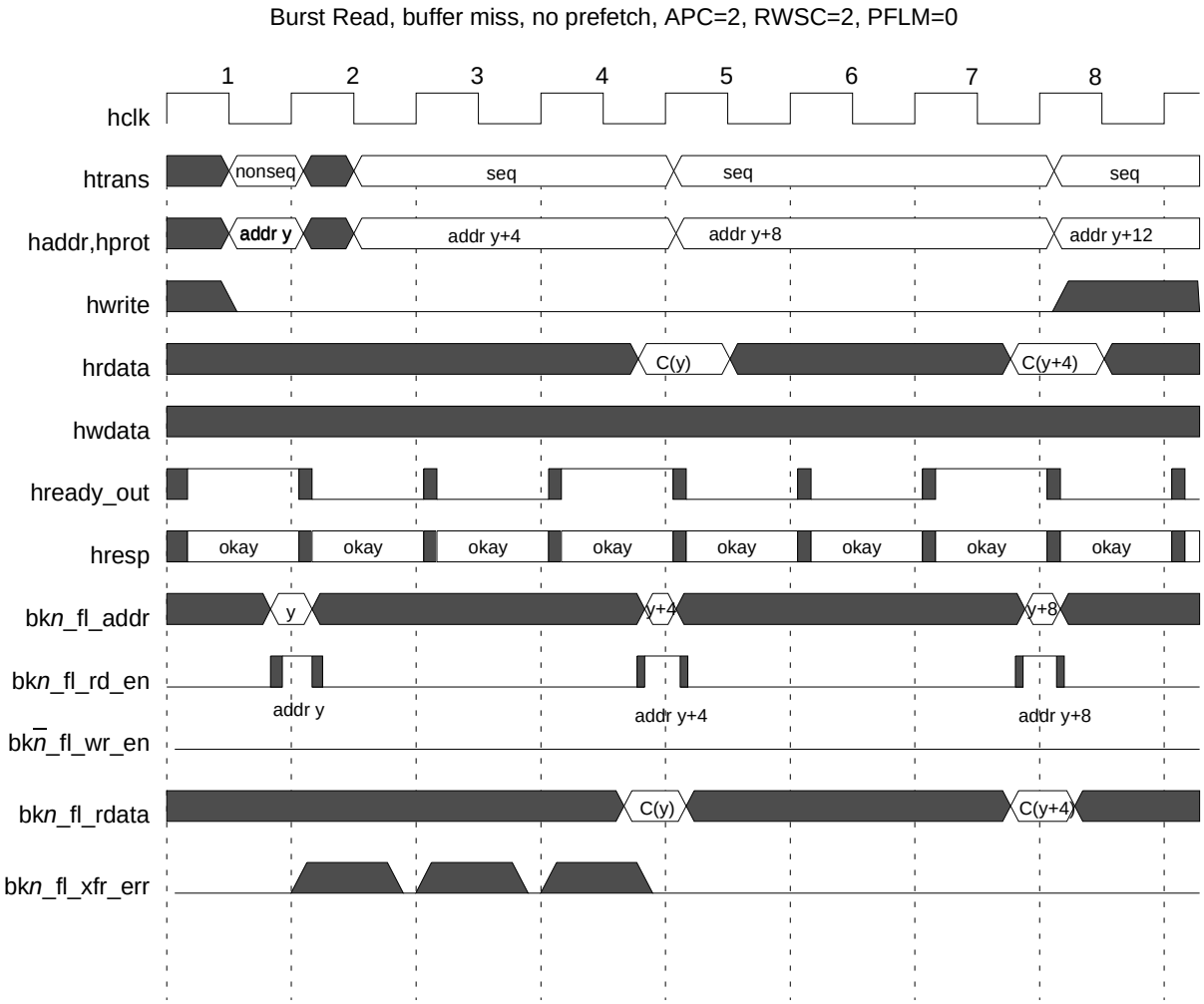


Figure 168. 3-cycle access, no prefetch, buffering disabled

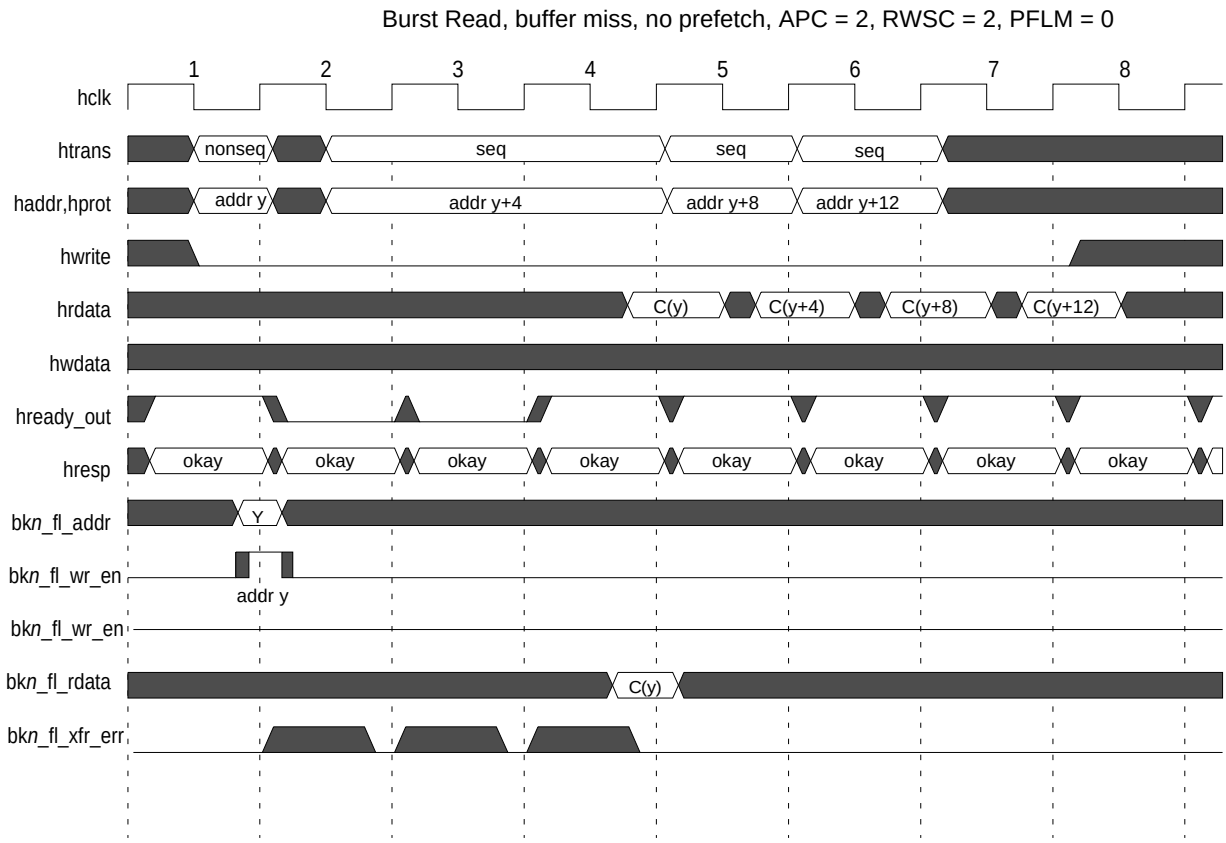


Figure 169. 3-cycle access, no prefetch, buffering enabled

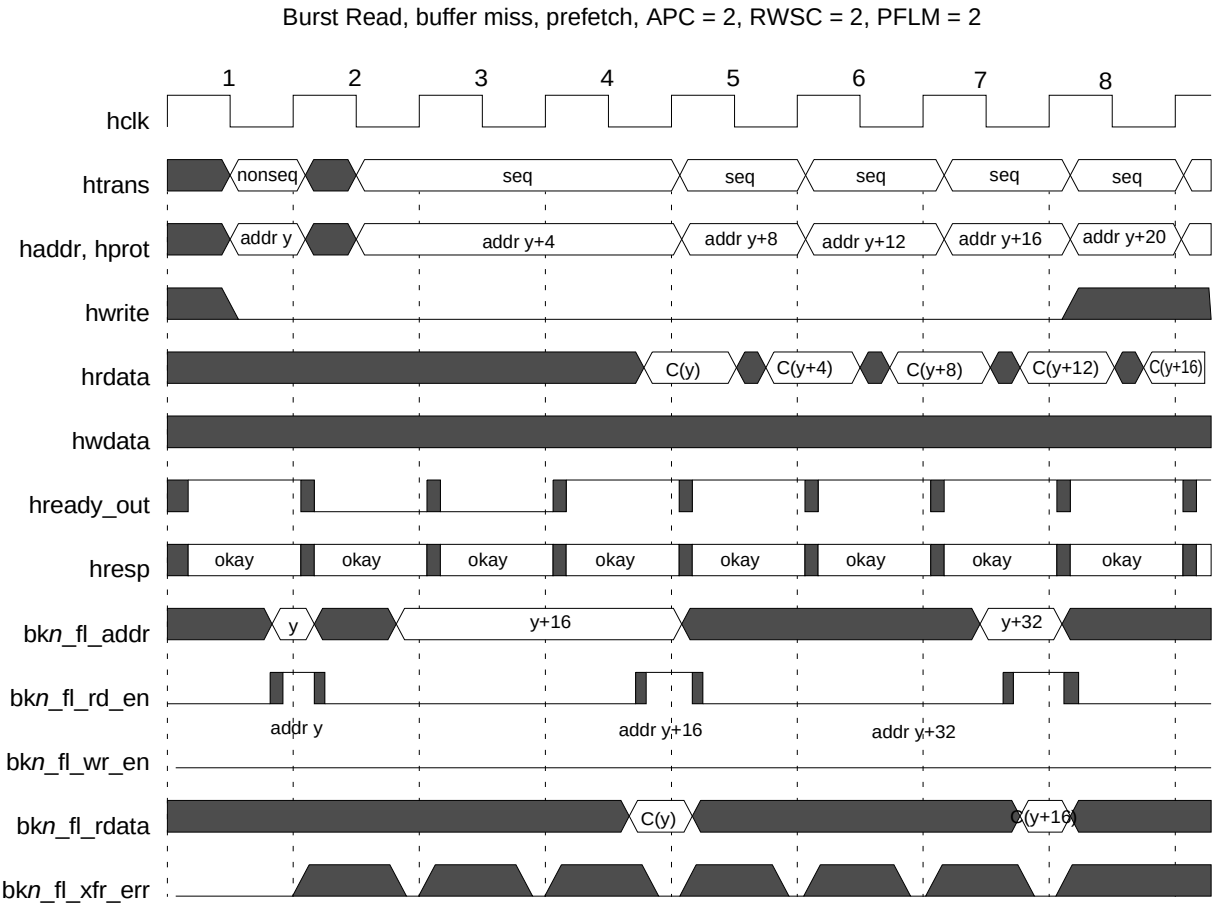


Figure 170. 3-cycle access, prefetch and buffering enabled

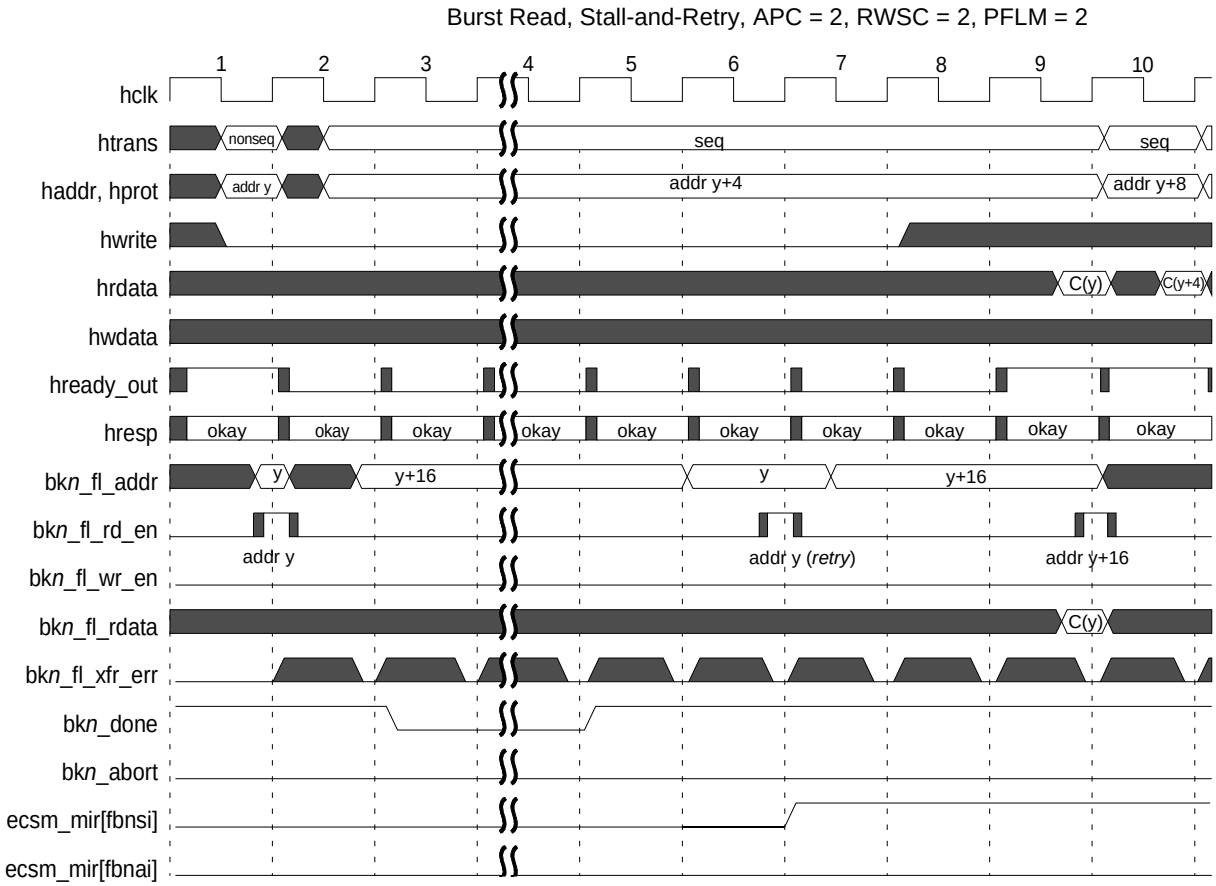


Figure 171. 3-cycle access, stall-and-retry with BK*n*_RWWC = 11*x*

As shown in *Figure 171*, the 3-cycle access to address y is interrupted when an operation causes the bk*n*_done signal to be negated, signaling that the array bank is busy with a high-voltage program or erase event. Eventually, this array operation completes (at the end of cycle 4) and bk*n*_done returns to a logical 1. In cycle 6, the platform Flash controller module retries the read to address y that was interrupted by the negation of bk*n*_done in cycle 3. Note that throughout cycles 2–9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y + 4 in the address phase. Depending on the state of the least-significant-bit of the BK*n*_RWWC control field, the hardware may also signal a stall notification interrupt (if BK*n*_RWWC = 110). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FB*n*SI] (Flash bank *n* stall interrupt).

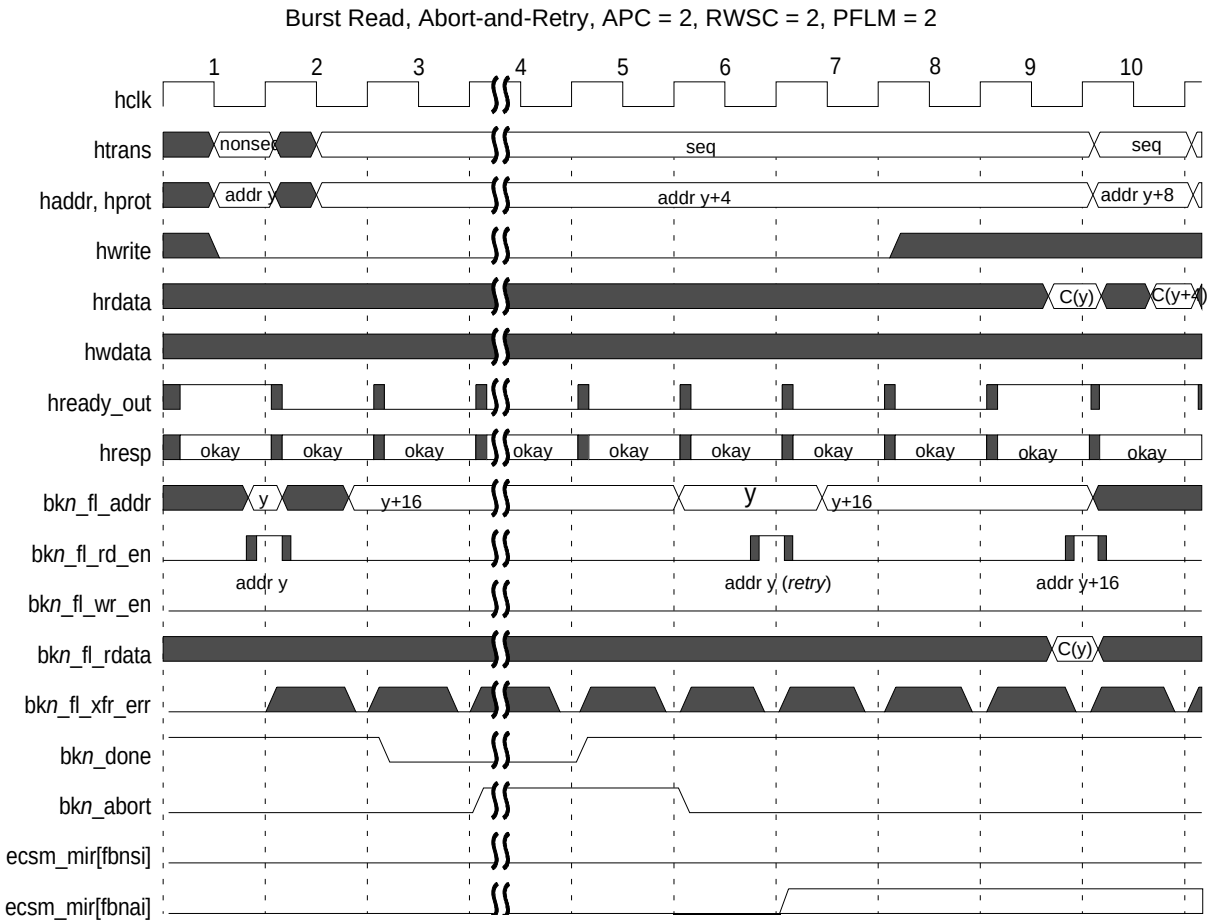


Figure 172. 3-cycle access, terminate-and-retry with BK*n*_RWWC = 10*x*

Figure 172 shows the terminate-while-write timing diagram. In this example, the 3-cycle access to address y is interrupted when an operation causes the bk*n*_done signal to be negated, signaling that the array bank is busy with a high-voltage program or erase event. Based on the setting of BK*n*_RWWC, once the bk*n*_done signal is detected as negated, the platform Flash controller asserts bk*n*_abort, which forces the Flash array to cancel the high-voltage program or erase event. The array operation completes (at the end of cycle 4) and bk*n*_done returns to a logical 1. It should be noted that the time spent in cycle 4 for *Figure 172* is considerably less than the time in the same cycle in *Figure 171* (because of the terminate operation). In cycle 6, the platform Flash controller module retries the read to address y that was interrupted by the negation of bk*n*_done in cycle 3. Note that throughout cycles 2–9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y+4 in the address phase. Depending on the state of the least-significant-bit of the BK*n*_RWWC control field, the hardware may also signal an termination notification interrupt (if BK*n*_RWWC = 100). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FB*n*AI] (Flash bank *n* termination interrupt).

17.3 Flash memory

17.3.1 Introduction

The Flash module provides electrically programmable and erasable non-volatile memory (NVM), which may be used for instruction and/or data storage.

The Flash module is arranged as two functional units: the Flash core and the memory interface.

The Flash core is composed of arrayed non-volatile storage elements, sense amplifiers, row decoders, column decoders and charge pumps. The arrayed storage elements in the Flash core are sub-divided into physically separate units referred to as blocks (or sectors).

The memory interface contains the registers and logic which control the operation of the Flash core. The memory interface is also the interface between the Flash module and a bus interface unit (BIU), and may contain the ECC logic and redundancy logic. The BIU connects the Flash module to a system bus.

The SPC56xP60/54 provides three Flash modules: two 512 KB code Flash modules, and one 64 KB data module.

17.3.2 Main features

- High read parallelism (128 bits)
- Error Correction Code (SEC-DED) to enhance data retention
- Double word program (64 bits)
- Sector erase
- Single bank architecture
  - Read-While-Modify not available within an individual module
  - Read-While-Modify can be performed between the two Flash modules
- Erase suspend available (program suspend not available)
- Software programmable program/erase protection to avoid unwanted writes
- Censored mode against piracy
- Shadow Sector available on code Flash module

17.3.3 Block diagram

Data Flash

The data Flash module contains one module, composed of a Single Bank: Bank 0, normally used for code storage. No Read-While-Modify operations are possible.

The Modify operations are managed by an embedded Flash Program/Erase Controller (FPEC). Commands to the FPEC are given through a user registers interface.

The read data bus is 32 bits wide, while the data Flash registers are on a separate 32-bit wide bus.

The high voltages needed for Program/Erase operations are internally generated. *Figure 173* shows the data Flash module structure.

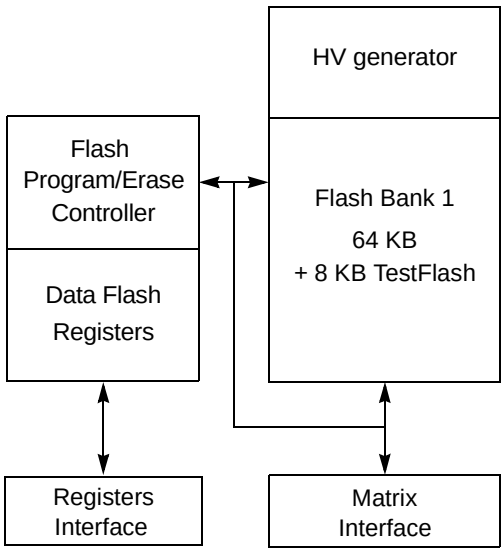


Figure 173. Data Flash module structure

Code Flash

The code Flash module contains the matrix modules normally used for Code storage. No Read-While-Modify operations are possible.

The Modify operations are managed by an embedded Flash Program/Erase Controller (FPEC). Commands to the FPEC are given through a User Registers Interface.

The read data bus is 2 × 128 bits wide, while the Flash registers are on a separate 32-bit wide bus.

The high voltages needed for Program/Erase operations are internally generated. *Figure 174* shows the code Flash module structure.

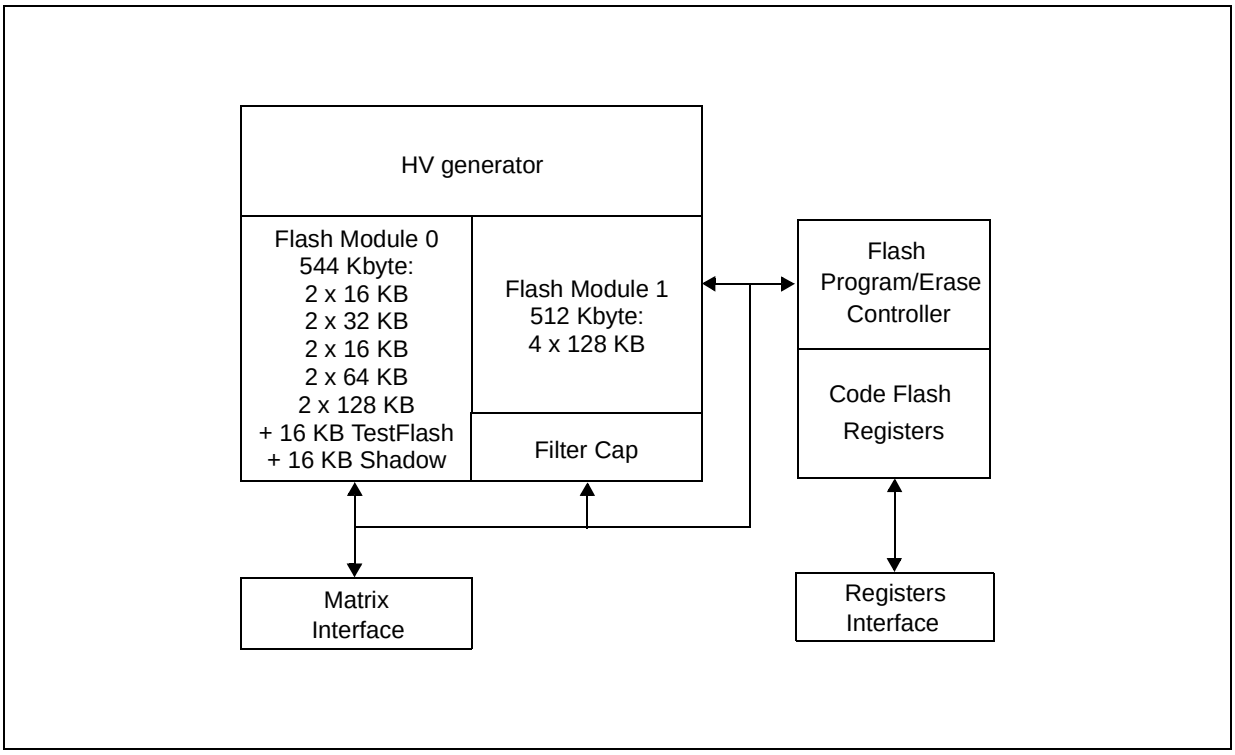


Figure 174. Code Flash module structure

17.3.4 Functional description

Macrocell structure

The Flash macrocell provides high density non-volatile memories with high-speed read access.

The Flash module is addressable by word (32 bits) or double-word (64 bits) for programming, and by page (128 bits) for reads. Reads done to the Flash always return 128 bits, although read page buffering may be done in the platform BIU.

Each read of the Flash module retrieves a page, or 4 consecutive words (128 bits) of information. The address for each word retrieved within a page differ from the other addresses in the page only by address bits (3:2).

The Flash page read architecture supports both cache and burst mode at the BIU level for high-speed read application.

The Flash module supports fault tolerance through Error Correction Code (ECC) and/or error detection. The ECC implemented within the Flash module will correct single bit failures and detect double bit failures.

The Flash module uses an embedded hardware algorithm implemented in the memory interface to program and erase the Flash core.

Control logic that works with the software block enables, and software lock mechanisms, is included in the embedded hardware algorithm to guard against accidental program/erase.

The hardware algorithm perform the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to guarantee data integrity and reliability.

A programmed bit in the Flash module reads as logic level 0 (or low).

An erased bit in the Flash module reads as logic level 1 (or high).

Program and erase of the Flash module requires multiple system clock cycles to complete.

The erase sequence may be suspended.

The program and erase sequences may be terminated.

Flash module sectorization

The code Flash module supports 1024 KB of user memory, plus 16 KB of test memory (a portion of which is one-time programmable by the user). An extra 16 KB sector is available as Shadow space usable for user option bits or censorship.

The Flash Multi-module is composed of three modules (0, 1 and 2): Read-While-Write is not supported.

The code Flash Bank 0 is divided in 18 sectors including a reserved sector named TestFlash, in which One Time Programmable (OTP) user data are stored, and a Shadow Sector in which user erasable configuration values can be stored (see *Table 169*).

768 KB devices use only the first 256 KB of Module0 plus the shadow block.

The data Flash Bank 1 is divided in five sectors including a reserved sector named TestFlash (see *Table 170*).

Table 169. Code Flash memory storage address map

Address	Use	Block(1), (2)	Size	SPC56AP54 768 KB	SPC56AP60 1 MB	Module
0x0000_0000	Low Address Space 256 KB	0	16 KB	Available	Available	0
0x0000_4000		1a	16 KB	Available	Available	
0x0000_8000		1b	32 KB	Available	Available	
0x0001_0000		2a	32 KB	Available	Available	
0x0001_8000		2b	16 KB	Available	Available	
0x0001_C000		3	16 KB	Available	Available	
0x0002_0000		4	64 KB	Available	Available	
0x0003_0000		5	64 KB	Available	Available	
0x0004_0000	Mid Address Space 256 KB	6	128 KB	Not available	Available	
0x0006_0000		7	128 KB	Not available	Available	
0x0008_0000	High Address Space 512 KB	8	128 KB	Available	Available	1
0x000A_0000		9	128 KB	Available	Available	
0x000C_0000		10	128 KB	Available	Available	
0x000E_0000		11	128 KB	Available	Available	
0x0010_0000	Reserved					
0x0020_0000	Shadow Address Space	B0SH	16 KB	Available	Available	0

Table 169. Code Flash memory storage address map (continued)

Address	Use	Block(1), (2)	Size	SPC56AP54 768 KB	SPC56AP60 1 MB	Module
0x0020_4000	Reserved					
0x0040_0000	Test Address Space	B0TF	16 KB	Available	Available	0
0x0040_4000	Reserved					

1. Boot can be performed from any block in the Low Address Space except blocks 1b and 2b.
2. Each block can be “Locked” against modification or “Selected” for erase.

Table 170. 64 KB data Flash module sectorization

Bank	Sector	Addresses	Size (KB)	Address space
B1	B1F0	0x0080_0000 to 0x0080_3FFF	16	Low Address Space
B1	B1F1	0x0080_4000 to 0x0080_7FFF	16	Low Address Space
B1	B1F2	0x0080_8000 to 0x0080_BFFF	16	Low Address Space
B1	B1F3	0x0080_C000 to 0x0080_FFFF	16	Low Address Space
B1	Reserved	0x0081_0000 to 0x00C0_1FFF	4040	Reserved
B1	B1TF	0x00C0_2000 to 0x00C0_3FFF	8	Test Address Space
B1	Reserved	0x00C0_4000 to 0x00FF_FFFF	4080	Reserved

Each Flash module is divided into blocks to implement independent program/erase protection. A software mechanism is provided to independently lock/unlock each block in address space against program and erase.

TestFlash block

The TestFlash block exists outside the normal address space and is programmed and read independently of the other blocks. The independent TestFlash block is included also to support systems that require non-volatile memory for security and/or to store system initialization information.

A section of the TestFlash is reserved to store the non-volatile information related to redundancy, configuration, and protection.

ECC is also applied to TestFlash. The usage of reserved TestFlash sectors is detailed in *Table 171*.

Table 171. TestFlash structure

Name	Description	Addresses		Size (bytes)
		Code TestFlash	Data TestFlash	
—	Reserved	0x0040_0000–0x0040_1FFF	0x00C0_0000– 0x00C0_1FFF	8192
—	Reserved	0x0040_2000–0x0040_3CFF	0x00C0_2000– 0x00C0_3CFF	7424
—	User Reserved	0x0040_3D00–0x0040_3DE7	0x00C0_3D00–0x00C0_3DE7	232

Table 171. TestFlash structure (continued)

Name	Description	Addresses		Size (bytes)
		Code TestFlash	Data TestFlash	
NVLML	Non-volatile Low/Mid address space block Locking register	0x0040_3DE8–0x0040_3DEF	0x00C0_3DE8–0x00C0_3DEF	8
NVHBL	Non-volatile High address space Block Locking register	0x0040_3DF0–0x0040_3DF7	0x00C0_3DF0–0x00C0_3DF7	8
NVSLL	Non-volatile Secondary Low/mid add space block Lock register	0x0040_3DF8–0x0040_3DFF	0x00C0_3DF8–0x00C0_3DFF	8
—	User Reserved	0x0040_3E00–0x0040_3EFF	0x00C0_3E00–0x00C0_3EFF	256
—	Reserved	0x0040_3F00–0x0040_3FFF	0x00C0_3F00–0x00C0_3FFF	256

Erase of the TestFlash block is always locked.

TestFlash block programming restrictions, in terms of how ECC is calculated, are similar to array programming restrictions. Only one program is allowed per 64-bit ECC segment.

Locations of the Code TestFlash block marked as reserved cannot be programmed by the user application. Locations of the Data TestFlash block marked as reserved cannot be programmed by the user application.

Shadow block

A Shadow block is present in each Code flash module, but not in the Data flash module. The Shadow block can be enabled by the BIU.

When the Shadow space is enabled, all the operations are mapped to the Shadow block.

User mode program and erase of the shadow block are enabled only when MCR[PEAS] is set.

The Shadow block may be locked/unlocked against program or erase by using the LML[TSLK] and SLL[STSLK] bitfields.

Program of the Shadow block has similar restriction as the array in terms of how ECC is calculated. Only one program is allowed per 64-bit ECC segment between erases.

Erase of the Shadow block is done similarly as an sectors erase.

The Shadow block contains specified data that are needed for user features.

The user area of Shadow block may be used for user-defined functions (possibly to store boot code, other configuration words, or factory process codes).

The usage of the Shadow sector is detailed in *Table 172*.

Table 172. Shadow sector structure

Name	Description	Addresses	Size (bytes)
—	User Area	0x0020_0000–0x0020_3DCF	15824
—	Reserved	0x0020_3DD0–0x0020_3DD7	8
NVPWD0–1	Non-volatile private censorship password 0–1 registers	0x0020_3DD8–0x0020_3DDF	8
NVSCI0–1	Non-volatile system censorship information 0–1 registers	0x0020_3DE0–0x0020_3DE7	8
—	Reserved	0x0020_3DE8–0x0020_3DFF	24
NVBIU2–3	Non-volatile bus interface unit 2–3 registers	0x0020_3E00–0x0020_3E0F	16
—	Reserved	0x0020_3E10–0x0020_3E17	8
NVUSRO	Non-volatile user options register	0x0020_3E18–0x0020_3E1F	8
—	Reserved	0x0020_3E20–0x0020_3FFF	480

17.3.5 Operating modes

The following operating modes are available in the Flash module:

- Reset
- User mode
- Low-power mode
- Power-down mode

Reset

A reset is the highest priority operation for the Flash module and terminates all other operations.

The Flash module uses reset to initialize registers and status bits to their default reset values.

If the Flash module is executing a program or erase operation (MCR[PGM] = 1 or MCR[ERS] = 1) and a reset is issued, the operation is suddenly terminated and the module disables the high voltage logic without damage to the high voltage circuits. Reset terminates all operations and forces the Flash module into User mode ready to receive accesses.

Reset and power-down must not be used as a systematic way to terminate a program or erase operation.

After reset is deasserted, read register access may be done, although it should be noted that registers that require updating from shadow information or other inputs may not read updated values until MCR[DONE] transitions. MCR[DONE] may be polled to determine if the Flash module has transitioned out of reset. Notice that the registers cannot be written until MCR[DONE] is high.

User mode

In User mode, the Flash module may be read, written (register writes and interlock writes), programmed, or erased.

The default state of the Flash module is the read state.

The main, shadow, and test address space can be read only in the read state.

The Flash registers are always available for reads. When the module is in power-down mode, most (but not all) registers are available for reads. The exceptions are documented.

The Flash module enters the read state on reset.

The module is in the read state under two sets of conditions:

- The read state is active when the module is enabled (User mode read).
- The read state is active when MCR[ERS] and MCR[ESUS] are set and MCR[PGM] is cleared (Erase Suspend).

No Read-While-Modify is available within an individual module, although one module can be read while the other is being written or otherwise modified.

Flash core reads return 128 bits (1 page = 2 double words).

Register reads return 32 bits (1 word).

Flash core reads are done through the BIU.

Register reads to unmapped register address space return all 0s.

Register writes to unmapped register address space have no effect.

Array reads attempted to invalid locations will result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes.

Interlock writes attempted to invalid locations, will result in an interlock occurring, but attempts to program these blocks will not occur since they are forced to be locked. Erase will occur to selected and unlocked blocks even if the interlock write is to an invalid location.

Simultaneous read cycles on the code Flash block and read/write cycles on the data Flash block are possible. However, simultaneous read/write accesses within a single block are not permitted.

Chip Select, Write Enable, addresses, and data input of registers are not internally latched and must be kept stable by the CPU for all the read/write access that lasts two clock cycles.

Low-power mode

The Low-power mode turns off most of the DC current sources within the Flash module.

The module (Flash core and registers) is not accessible for read or write operations once it has entered Low-power mode.

Wake-up time from Low-power mode is faster than wake-up time from Power-down mode.

The user may not read some registers (UMISR0–4, UT1–2 and part of UT0) until the Low-power mode is exited. Write access is locked on all the registers in Low-power mode.

When exiting from Low-power mode, the Flash module returns to its previous state in all cases, unless it was in the process of executing an erase high voltage operation at the time of entering Low-power mode.

If the Flash module is put into Low-power mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation when the Flash module exits from Low-power mode by clearing MCR[ESUS]. MCR[EHV] must be set to resume the erase operation.

If the Flash module is put in Low-power mode during a program operation, the operation is completed in all cases. Low-power mode is entered only after the programming ends.

Power-down mode cannot be entered when Low-power mode is active. The module must first be set to User mode (or reset) when cycling between Low-power mode and Power-down mode.

Power-down mode

The Power-down mode allows turning off all Flash DC current sources so that power dissipation is limited only to leakage in this mode.

In Power-down mode, no reads from or writes to the Flash are possible.

When enabled, the Flash module returns to its previous state in all cases, unless in the process of executing an erase high voltage operation at the time of entering Power-down mode. If the Flash module is put into Power-down mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation when the Flash module exits Power-down mode by clearing the MCR[ESUS] bit. MCR[EHV] must be set to resume the erase operation.

If the Flash module is placed in Power-down mode during a program operation, the operation will be completed in any case and the Power-down mode is entered only after the programming is completed.

If the Flash module is put in Power-down mode and the vector table remains mapped in the Flash address space, the user must observe than the Flash module will require a longer interrupt response time. This should be accomplished by adding several wait states.

Low-power mode cannot be entered when Power-down mode is active. The module must first be set to User mode (or reset) when cycling between Low-power mode and Power-down mode.

17.3.6 Registers description

The Flash user registers mapping is shown in *Table 173*. Except as noted, registers and offsets are identical for the code Flash and data Flash blocks.

Table 173. Flash registers

Offset from xxxx_BASE (0xC3F8_8000)	Register	Location
0x0000		*on page 396*
0x0004	Low/mid Address Space Block Locking Register (LML)	*on page 401*
0x0008	High Address Space Block Locking Register (HBL)	*on page 403*
0x000C	Secondary Low/mid Address Space Block Lock Register (SLL)	*on page 405*
0x0010	Low/mid Address Space Block Select Register (LMS)	*on page 408*
0x0014	High Address Space Block Select Register (HBS)	*on page 409*
0x0018	Address Register (ADR)	*on page 410*
0x001C	Platform Flash Configuration Register 0 (PFCR0)(1)	*on page 411*
0x0020	Platform Flash Configuration Register 1 (PFCR1)(1)	*on page 416*
0x0024	Platform Flash Access Protection Register (PFAPR)(1)	*on page 418*
0x0028	Bus Interface Unit reg 3 (BIU3)(1)	*on page 420*
0x003C	User Test Register 0 (UT0)	*on page 421*
0x0040	User Test Register 1 (UT1)	*on page 423*
0x0044	User Test Register 2 (UT2)(1)	*on page 423*
0x0048	User Multiple Input Signature Register 0 (UMISR0)	*on page 424*
0x004C	User Multiple Input Signature Register 1 (UMISR1)	*on page 425*
0x0050	User Multiple Input Signature Register 2 (UMISR2)(1)	*on page 425*
0x0054	User Multiple Input Signature Register 3 (UMISR3)(1)	*on page 426*
0x0058	User Multiple Input Signature Register 4 (UMISR4)(1)	*on page 427*
0x005C–0x3FFF	Reserved	

1. This register is not implemented on the data Flash block.

17.3.7 Register map

Table 174. Flash 1056 KB multi-module register map

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x00	MCR	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
		EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV

Table 174. Flash 1056 KB multi-module register map (continued)

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x04	LML	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
		LLK15	LLK14	LLK13	LLK12	LLK11	LLK10	LLK9	LLK8	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
0x08	HBL	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	HLK3	HLK2	HLK1	HLK0
0x0C	SLL	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
		SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
0x10	LMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL1	MSL0
		LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
0x14	HBS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	HSL3	HSL2	HSL1	HSL0
0x18	ADR	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
		AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
0x1C	PFCR0	BI031	BI030	BI029	BI028	BI027	BI026	BI025	BI024	BI023	BI022	BI021	BI020	BI019	BI018	BI017	BI016
		BI015	BI014	BI013	BI012	BI011	BI010	BI009	BI008	BI007	BI006	BI005	BI004	BI003	BI002	BI001	BI000
0x20	PFCR1	BI131	BI130	BI129	BI128	BI127	BI126	BI125	BI124	BI123	BI122	BI121	BI120	BI119	BI118	BI117	BI116
		BI115	BI114	BI113	BI112	BI111	BI110	BI109	BI108	BI107	BI106	BI105	BI104	BI103	BI102	BI101	BI100
0x24	PFAPR	BI231	BI230	BI229	BI228	BI227	BI226	BI225	BI224	BI223	BI222	BI221	BI220	BI219	BI218	BI217	BI216
		BI215	BI214	BI213	BI212	BI211	BI210	BI209	BI208	BI207	BI206	BI205	BI204	BI203	BI202	BI201	BI200
0x28	BIU3	BI331	BI330	BI329	BI328	BI327	BI326	BI325	BI324	BI323	BI322	BI321	BI320	BI319	BI318	BI317	BI316
		BI315	BI314	BI313	BI312	BI311	BI310	BI309	BI308	BI307	BI306	BI305	BI304	BI303	BI302	BI301	BI300
0x28	Reserved																
0x3C	UT0	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
		0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
0x40	UT1	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
		DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
0x44	UT2	DAI63	DAI62	DAI61	DAI60	DAI59	DAI58	DAI57	DAI56	DAI55	DAI54	DAI53	DAI52	DAI51	DAI50	DAI49	DAI48
		DAI47	DAI46	DAI45	DAI44	DAI43	DAI42	DAI41	DAI40	DAI39	DAI38	DAI37	DAI36	DAI35	DAI34	DAI33	DAI32
0x48	UMISR0	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
		MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
0x4C	UMISR1	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
		MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
0x50	UMISR2	MS095	MS094	MS093	MS092	MS091	MS090	MS089	MS088	MS087	MS086	MS085	MS084	MS083	MS082	MS081	MS080
		MS079	MS078	MS077	MS076	MS075	MS074	MS073	MS072	MS071	MS070	MS069	MS068	MS067	MS066	MS065	MS064

Table 174. Flash 1056 KB multi-module register map (continued)

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x54	UMISR3	MS127	MS126	MS125	MS124	MS123	MS122	MS121	MS120	MS119	MS118	MS117	MS116	MS115	MS114	MS113	MS112
		MS111	MS110	MS109	MS108	MS107	MS106	MS105	MS104	MS103	MS102	MS101	MS100	MS099	MS098	MS097	MS096
0x58	UMISR4	MS159	MS158	MS157	MS156	MS155	MS154	MS153	MS152	MS151	MS150	MS149	MS148	MS147	MS146	MS145	MS144
		MS143	MS142	MS141	MS140	MS139	MS138	MS137	MS136	MS135	MS134	MS133	MS132	MS131	MS130	MS129	MS128

Table 175. Flash 64 KB bank1 register map

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x00	MCR	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
		EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
0x04	LML	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
		LLK15	LLK14	LLK13	LLK12	LLK11	LLK10	LLK9	LLK8	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
0x08	Reserved																
0x0C	SLL	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
		SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
0x10	LMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL1	MSL0
		LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
0x14	Reserved																
0x18	ADR	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
		AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
0x1C–0x3B	Reserved																
0x3C	UT0	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
		0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
0x40	UT1	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
		DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
0x44–0x47	Reserved																
0x48	UMISR0	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
		MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
0x4C	UMISR1	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
		MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
0x50–0x5B	Reserved																

In the following sections, some non-volatile registers are described. Please notice that such entities are not Flip-Flops, but locations of TestFlash or Shadow sectors with a special meaning.

During the Flash initialization phase, the FPEC reads these non-volatile registers and updates their related volatile registers. When the FPEC detects ECC double errors in these special locations, it behaves in the following way:

- In case of a failing system locations (configurations, device options, redundancy, EmbAlgo firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, censorship, BIU, ...), the volatile registers are filled with all 1s and the Flash initialization ends by clearing the MCR[PEG] bit.

Module Configuration Register (MCR)

The Module Configuration Register enables and monitors all the modify operations of each Flash module. Identical MCRs are provided in the code Flash and the data Flash blocks.

Figure 175. Module Configuration Register (MCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
W	r1c															
Reset	0	0	0	0	0	—(1)	—(1)	—(1)	0	—(1)	1	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	r1c	r1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

1. The value for this bit is different between the code and data Flash modules. See the bitfield description.

Table 176. MCR field descriptions

Field	Description
EDC 0	ECC Data Correction EDC provides information on previous reads. If a ECC Single Error detection and correction occurs, the EDC bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. In the event of a ECC Double Error detection, this bit is not set. If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally. 1 An ECC Single Error occurred and was corrected during a previous read.
1:4	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 176. MCR field descriptions (continued)

Field	Description
SIZE[2:0] 5:7	Array space SIZE 2–0 The value of SIZE field depends on the size of the Flash module: 000 128 KB 001 256 KB 010 512 KB 011 1056 KB (the value for the SPC56xP60/54 device in the code Flash module) 100 Reserved (1536 KB) 101 Reserved (2048 KB) 110 64 KB (the value for the device in the data Flash module) 111 Reserved The value for this bitfield is different between the code and data Flash modules.
8	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
LAS[2:0] 9:11	Low Address Space 2–0 The value of the LAS field corresponds to the configuration of the Low Address Space: 000 Reserved 001 2 × 128 KB 010 32 KB + (2 × 16 KB) + (2 × 32 KB) + 128 KB (the value for the SPC56xP60/54device in the code Flash module) 011 Reserved 100 Reserved 101 Reserved 110 4 × 16 KB (the value for the SPC56xP60/54device in the data Flash module) 111 (2 × 16 KB) + (2 × 32 KB) + (2 × 16 KB) + (2 × 64 KB) The value for this bitfield is different between the code and data Flash modules.
12:14	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
MAS 15	Mid Address Space The value of the MAS field corresponds to the configuration of the Mid Address Space: 0 0 KB or 2 × 128 KB 1 Reserved
EER 16	ECC Event Error EER provides information on previous reads. When an ECC Double Error detection occurs, the EER bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. In the event of a ECC Single Error detection and correction, this bit will not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally. 1 An ECC Double Error occurred during a previous read.

Table 176. MCR field descriptions (continued)

Field	Description
RWE 17	Read-while-Write event Error RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit is set to 1. Read-While-Write Error means that a read access to the Flash module has occurred while the FPEC was performing a program or Erase operation or an Array Integrity Check. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally. 1 A RWW Error occurred during a previous read. If stall/terminate-while-write is used, the software should ignore the setting of the RWE flag and should clear this flag after each erase operation. If stall/terminate-while-write is not used, software can handle the RWE error normally.
18:19	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
PEAS 20	Program/Erase Access Space PEAS indicates which space is valid for program and Erase operations: main array space or shadow/test space. PEAS = 0 indicates that the main address space is active for all Flash module program and erase operations. PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for Modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes). 0 Shadow/Test address space is disabled for program/erase and main address space enabled. 1 Shadow/Test address space is enabled for program/erase and main address space disabled.
DONE 21	Modify Operation Done DONE indicates if the Flash module is performing a high voltage operation. DONE is set to 1 on termination of the Flash module reset. DONE is cleared to 0 just after a 0-to-1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation. DONE is set to 1 at the end of program and erase high voltage sequences. DONE is set to 1 (within t_{PABT} or t_{EABT}, equal to P/E Abort Latency) after a 1-to-0 transition of EHV, which terminates a high voltage program/erase operation. DONE is set to 1 (within t_{ESUS}, time equal to Erase Suspend Latency) after a 0-to-1 transition of ESUS, which suspends an erase operation. 0 Flash is executing a high voltage operation. 1 Flash is not executing a high voltage operation.

Table 176. MCR field descriptions (continued)

Field	Description
PEG 22	Program/Erase Good The PEG bit indicates the completion status of the last Flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the Flash module is reset, unless a Flash initialization error has been detected. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to a termination or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1-to-0 transition or EHV makes a 0-to-1 transition. The value in PEG is not valid after a 0-to-1 transition of DONE caused by ESUS being set to logic 1. If program or erase are attempted on blocks that are locked, the response is PEG = 1, indicating that the operation was successful, and the content of the block were properly protected from the program or erase operation. If a program operation tries to program at 1 bits that are at 0, the program operation is correctly executed on the new bits to be programmed at 0, but PEG is cleared, indicating that the requested operation has failed. In Array Integrity Check or Margin Mode, PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS[0:1]. 0 Program or erase operation failed; or program, erase, Array Integrity Check, or Margin Mode was terminated. 1 Program or erase operation successful; or Array Integrity Check or Margin Mode completed successfully.
23:26	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
PGM 27	Program PGM sets up the Flash module for a program operation. A 0-to-1 transition of PGM initiates a program sequence. A 1-to-0 transition of PGM ends the program sequence. PGM can be set only under User mode Read (ERS is low and UT0[AIE] is low). PGM can be cleared by the user only when EHV is low and DONE is high. PGM is cleared on reset. 0 Flash is not executing a program sequence. 1 Flash is executing a program sequence.
PSUS 28	Program Suspend Writing to this bit has no effect, but the written data can be read back.
ERS 29	Erase ERS sets up the Flash module for an Erase operation. A 0-to-1 transition of ERS initiates an Erase sequence. A 1-to-0 transition of ERS ends the Erase sequence. ERS can be set only under User mode Read (PGM is low and UT0[AIE] is low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0 Flash is not executing an Erase sequence. 1 Flash is executing an Erase sequence.

Table 176. MCR field descriptions (continued)

Field	Description
ESUS 30	Erase Suspend ESUS indicates that the Flash module is in Erase Suspend or in the process of entering a Suspend state. The Flash module is in Erase Suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS = 1 and EHV = 1, and PGM = 0. A 0-to-1 transition of ESUS starts the sequence that sets DONE and places the Flash in erase suspend. The Flash module enters Suspend within t_{ESUS} of this transition. ESUS can be cleared only when DONE = 1 and EHV = 1, and PGM = 0. A 1-to-0 transition of ESUS with EHV = 1 starts the sequence that clears DONE and returns the Module to Erase. The Flash module cannot exit Erase Suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.
EHV 31	Enable High Voltage The EHV bit enables the Flash module for a high voltage program/Erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/Erase sequence. EHV may be set under one of the following conditions: – Erase (ERS = 1, ESUS = 0, UT0[AIE] = 0) – Program (ERS = 0, ESUS = 0, PGM = 1, UT0[AIE] = 0) In normal operation, a 1-to-0 transition of EHV with DONE high and ESUS low terminates the current program/Erase high voltage operation. When an operation is terminated, there is a 1-to-0 transition of EHV with DONE low and the eventual Suspend bit low. A termination causes the value of PEG to be cleared, indicating a failing program/Erase; address locations being operated on by the terminated operation contain indeterminate data after a termination. A suspended operation cannot be terminated. Terminating a high voltage operation leaves the Flash module addresses in an indeterminate data state. This may be recovered by executing an Erase on the affected blocks. EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low. 0 Flash is not enabled to perform an high voltage operation. 1 Flash is enabled to perform an high voltage operation.

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The Flash module does not allow the user to write bits simultaneously which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. *Table 177* shows the bit changing priorities.

Table 177. MCR bits set/clear priority levels

Priority level	MCR bits
1	ERS
2	PGM

Table 177. MCR bits set/clear priority levels (continued)

Priority level	MCR bits
3	EHV
4	ESUS

If the user attempts to write two or more MCR bits simultaneously, only the bit with the lowest priority level is written.

Low/Mid Address Space Block Locking register (LML)

The Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the SLL register, determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status. Identical LML registers are provided in the code Flash and the data Flash blocks.

In the code Flash module, the LML register has a related Non-Volatile Low/Mid Address Space Block Locking register (NVLML) located in TestFlash that contains the default reset value for LML. The NVLML register is read during the reset phase of the Flash module and loaded into the LML. The reset value is 0x00XX_XXXX, initially determined by the NVLML value from test sector.

Figure 176. Low/Mid Address Space Block Locking register (LML)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LLK 15	LLK 14	LLK 13	LLK 12	LLK 11	LLK 10	LLK 9	LLK 8	LLK 7	LLK 6	LLK 5	LLK 4	LLK 3	LLK 2	LLK 1	LLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Non-Volatile Low/Mid Address Space Block Locking register (NVLML)

Figure 177. Non-Volatile Low/Mid Address Space Block Locking register (NVLML)

Address: Base + 0x40_3DE8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LLK 15	LLK 14	LLK 13	LLK 12	LLK 11	LLK 10	LLK 9	LLK 8	LLK 7	LLK 6	LLK 5	LLK 4	LLK 3	LLK 2	LLK 1	LLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

The NVLML register is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes. Identical NVLML registers are provided in the code Flash and the data Flash blocks.

Table 178. LML and NVLML field descriptions

Field	Description
LME(1) 0	Low/Mid Address Space Block Enable This bit enables the Lock registers (TSLK, MLK[1:0], and LLK[15:0]) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the LML register. 0 Low Address Locks are disabled: TSLK, MLK[1:0], and LLK[15:0] cannot be written. 1 Low Address Locks are enabled: TSLK, MLK[1:0], and LLK[15:0] can be written.
1:10	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
TSLK 11	Test/Shadow Address Space Block Lock This bit locks the block of Test and Shadow Address Space from program and Erase (Erase is any case forbidden for Test block). A value of 1 in the TSLK register signifies that the Test/Shadow block is locked for program and Erase. A value of 0 in the TSLK register signifies that the Test/Shadow block is available to receive program and Erase pulses. The TSLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high. 0 Test/Shadow Address Space Block is unlocked and can be modified (if also SLL[STSLK] = 0). 1 Test/Shadow Address Space Block is locked and cannot be modified.
12:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 178. LML and NVLML field descriptions (continued)

Field	Description
MLK[1:0] 14:15	Mid Address Space Block Lock 1-0 These bits lock the blocks of Mid Address Space from program and Erase. MLK[1:0] are related to sectors B0F[7:6], respectively. See *Table 169* for more information. A value of 1 in a bit of the MLK bitfield signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the MLK bitfield signifies that the corresponding block is available to receive program and Erase pulses. The MLK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK bitfield is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the MLK bitfields. The MLK bits may be written as a register. Reset causes the bits to revert to their TestFlash block value. The default value of the MLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the MLK bits default to locked, and are not writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. MLK is not writable unless LME is high. 0 Mid Address Space Block is unlocked and can be modified (if also SLL[SMLK] = 0). 1 Mid Address Space Block is locked and cannot be modified.
LLK[15:0] 16:31	Low Address Space Block Lock 15-0 These bits lock the blocks of Low Address Space from program and Erase. For code Flash, LLK[5:0] are related to sectors B0F[5:0], respectively. See *Table 169* for more information. For data Flash, LLK[3:0] are related to sectors B1F[3:0], respectively. See *Table 170* for more information. A value of 1 in a bit of the LLK bitfield signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the LLK bitfield signifies that the corresponding block is available to receive program and Erase pulses. The LLK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK bitfield is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK bitfields. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value is always 1 (independent of the TestFlash block), and register writes have no effect. In the code Flash macrocell, bits LLK[15:8] are read-only and locked at 0. In the data Flash macrocell, bits LLK[15:4] are read-only and locked at 1. LLK is not writable unless LME is high. 0 Low Address Space Block is unlocked and can be modified (if also SLL[SLK] = 0). 1 Low Address Space Block is locked and cannot be modified.

1. This field is present only in LML

High Address Space Block Locking register (HBL)

The High Address Space Block Locking register provides a means to protect blocks from being modified.

The HBL register has a related Non- Volatile High Address Space Block Locking register (NVHBL) located in TestFlash that contains the default reset value for HBL. The reset value is 0x0000_000X, initially determined by NVHBL.

The NVHBL register is read during the reset phase of the Flash module and loaded into the HBL.

Figure 178. High Address Space Block Locking register (HBL)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	HLK3	HLK2	HLK1	HLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x

Non-Volatile High Address Space Block Locking register (NVHBL)

The NVHBL register is a 64-bit register, the 32 most significant bits of which (bits 63–32) “don’t-care” bits that are eventually used to manage ECC codes.

Figure 179. Non-Volatile High Address Space Block Locking register (NVHBL)

Address: 0x40_3DF0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	HLK3	HLK2	HLK1	HLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x

Table 179. HBL and NVHBL field descriptions

Field	Description
HBE 0	High address space Block Enable This bit enables the Lock bits (HLK[3:0]) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B2_2222 must be written to the HBL register. 0 High Address Locks are disabled: HLK[3:0] cannot be written. 1 High Address Locks are enabled: HLK[3:0] can be written.
1:25	*Reserved* (Read Only). A write to these bits has no effect. A read of these bits always outputs 0.

Table 179. HBL and NVHBL field descriptions (continued)

Field	Description
HLK[3:0] 28:31	High Address Space Block Lock 3–0 These bits lock the blocks of High Address Space from program and Erase. Not all the HLK[3:0] are used for this memory cut that is all mapped in mid and low address space. A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive program and Erase pulses. The HLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the HLK registers. The HLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the HLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the HLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the 512 KB Flash macrocell, bits HLK[3:0] are read-only and locked at 1. HLK is not writable unless HBE is high. 0 High Address Space Block is unlocked and can be modified. 1 High Address Space Block is locked and cannot be modified.

Secondary Low/Mid Address Space Block Locking register (SLL)

The Secondary Low/Mid Address Space Block Locking register provides an alternative means to protect blocks from being modified. These bits, along with bits in the LML register, determine if the block is locked from program or Erase. An “OR” of LML and SLL determine the final lock status. Identical SLL registers are provided in the code Flash and the data Flash blocks.

In the code Flash module, the SLL register has a related Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL) located in TestFlash that contains the default reset value for SLL. The reset value is 0x00XX_XXXX, initially determined by NVSLL.

The NVSLL register is read during the reset phase of the Flash module and loaded into the SLL.

Figure 180. Secondary Low/mid address space block Locking reg (SLL)

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	STS LK	0	0	SMK 1	SMK 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK 15	SLK 14	SLK 13	SLK 12	SLK 11	SLK 10	SLK 9	SLK 8	SLK 7	SLK 6	SLK 5	SLK 4	SLK 3	SLK 2	SLK 1	SLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL)

The NVSLL register is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes. Identical NVSLL registers are provided in the code Flash and the data Flash blocks.

Figure 181. Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL)

Address: Base + 0x40_3DF8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 180. SLL and NVSLL field descriptions

Field	Description
SLE[1] 0	Secondary Low/Mid Address Space Block Enable This bit enables the Lock registers (STSLK, SMK[1:0], and SLK[15:0]) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C3_3333 must be written to the SLL register. 0 Secondary Low/Mid Address Locks are disabled: STSLK, SMK[1:0], and SLK[15:0] cannot be written. 1 Secondary Low/Mid Address Locks are enabled: STSLK, SMK[1:0], and SLK[15:0] can be written.
1:10	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
STSLK 11	Secondary Test/Shadow address space block LocK This bit is used as an alternate means to lock the block of Test and Shadow Address Space from program and Erase (Erase is any case forbidden for Test block). A value of 1 in the STSLK bitfield signifies that the Test/Shadow block is locked for program and Erase. A value of 0 in the STSLK register signifies that the Test/Shadow block is available to receive program and Erase pulses. The STSLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high. 0 Test/Shadow Address Space Block is unlocked and can be modified (if also LML[TSLK] = 0). 1 Test/Shadow Address Space Block is locked and cannot be modified.

Table 180. SLL and NVSLL field descriptions (continued)

Field	Description
12:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
SMK[1:0] 14:15	Secondary Mid Address Space Block Lock 1–0 These bits are used as an alternate means to lock the blocks of Mid Address Space from program and Erase. SMK[1:0] are related to sectors B0F[7:6], respectively. See *Table 169* for more information. A value of 1 in a bit of the SMK register signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the SMK register signifies that the corresponding block is available to receive program and Erase pulses. The SMK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes have no effect. SMK is not writable unless SLE is high. 0 Mid Address Space Block is unlocked and can be modified (if also LML[MLK] = 0). 1 Mid Address Space Block is locked and cannot be modified.
SLK[15:0] 16:31	Secondary Low Address Space Block Lock 15–0 These bits are used as an alternate means to lock the blocks of Low Address Space from program and Erase. For code Flash, SLK[5:0] are related to sectors B0F[5:0], respectively. See *Table 169* for more information. For data Flash, SLK[3:0] are related to sectors B1F[3:0], respectively. See *Table 170* for more information. A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and Erase pulses. The SLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset causes the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SLK bits default to locked, and are not writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the code Flash macrocell, bits SLK[15:8] are read-only and locked at 0. In the data Flash macrocell, bits SLK[15:4] are read-only and locked at 1. SLK is not writable unless SLE is high. 0 Low Address Space Block is unlocked and can be modified (if also LML[LLK] = 0). 1 Low Address Space Block is locked and cannot be modified.

1. This field is present only in SLL

Low/Mid Address Space Block Select register (LMS)

The Low/Mid Address Space Block Select register provides a means to select blocks to be operated on during erase. Identical LMS registers are provided in the code Flash and the data Flash blocks.

Figure 182. Low/Mid Address Space Block Select register (LMS)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL 1	MSL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LSL 15	LSL 14	LSL 13	LSL 12	LSL 11	LSL 10	LSL 9	LSL 8	LSL 7	LSL 6	LSL 5	LSL 4	LSL 3	LSL 2	LSL 1	LSL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 181. LMS field descriptions

Field	Description
0:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
MSL[1:0] 14:15	Mid Address Space Block Select 1–0 A value of 1 in the select register signifies that the block is selected for erase. See *Table 169* for more information. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. MSL[1:0] are related to sectors B0F[7:6], respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding MSL bits default to unselected, and are not writable. The reset value will always be 0, and register writes have no effect. 0 Mid Address Space Block is unselected for Erase. 1 Mid Address Space Block is selected for Erase.

Table 181. LMS field descriptions (continued)

Field	Description
LSL[15:0] 16:31	Low Address Space Block Select 15–0 A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. For code Flash, LSL[5:0] are related to sectors B0F[5:0], respectively. See *Table 169* for more information. For data Flash, LSL[3:0] are related to sectors B1F[3:0], respectively. See *Table 170* for more information. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. In the code Flash macrocell, bits LSL[15:6] are read-only and locked at 0. In the data Flash macrocell, bits LSL[15:4] are read-only and locked at 0. 0 Low Address Space Block is unselected for Erase. 1 Low Address Space Block is selected for Erase.

High Address Space Block Select register (HBS)

The High Address Space Block Select register provides a means to select blocks to be operated on during erase.

Figure 183. High Address Space Block Select register (HBS)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	HSL3	HSL2	HSL1	HSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 182. HBS field descriptions

Field	Description
0:25	*Reserved* (Read Only). A write to these bits has no effect. A read of these bits always outputs 0.

Table 182. HBS field descriptions (continued)

Field	Description
HSL[3:0] 26:31	High Address Space Block Select 3–0 A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. All the HSL[3:0] are not used for this memory cut that is all mapped in mid and low address space. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding HSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. In the 1056 KB Flash macrocell, bits HSL[3:0] are read-only and locked at 0. 0 High Address Space Block is unselected for Erase. 1 High Address Space Block is selected for Erase.

Address Register (ADR)

The Address Register provides the first failing address in the event module failures (ECC, RWW, or FPEC) or the first address at which a ECC single error correction occurs.

Figure 184. Address Register (ADR)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD 15	AD 14	AD 13	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 183. ADR field descriptions

Field	Description
0:8	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 183. ADR field descriptions (continued)

Field	Description
AD[22:3] 9:28	Address 22–3 ADR provides the first failing address in the event of ECC error (MCR[EER] set) or the first failing address in the event of RWW error (MCR[RWE] set), or the address of a failure that may have occurred in a FPEC operation (MCR[PEG] cleared). ADR also provides the first address at which a ECC single error correction occurs (MCR[EDC] set). The ECC double error detection takes the highest priority, followed by the RWW error, the FPEC error, and the ECC single error correction. When accessed ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in *Table 184*. This address is always a double word address that selects 64 bits. In case of a simultaneous ECC double error detection on both double words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC single error correction on both double words of the same page. In User mode, ADR is read only.
29:31	*Reserved* Write these bits has no effect and read these bits always outputs 0.

Table 184. ADR content: priority list

Priority level	Error flag	ADR content
1	MCR[EER] = 1	Address of first ECC Double Error
2	MCR[RWE] = 1	Address of first RWW Error
3	MCR[PEG] = 0	Address of first FPEC Error
4	MCR[EDC] = 1	Address of first ECC Single Error Correction

Platform Flash Configuration Register 0 (PFCR0)

The Platform Flash Configuration Register 0 (PFCR0) defines the configuration associated with Flash memory bank0, which corresponds to the code Flash. The register is described in *Figure 185* and *Table 185*.

Note: This register is not implemented on the data Flash block.

Figure 185. Platform Flash Configuration Register 0 (PFCR0)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BK0_APC				BK0_WWSC				BK0_RWSC							BK0_RWWC2
Reset	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BK0_RWWC1	B0_P1_BCFG		B0_P1_DPFE	B0_P1_IPFE	B0_P1_PFLM		B0_P1_BFE	BK0_RWWC0	B0_P0_BCFG		B0_P0_DPFE	B0_P0_IPFE	B0_P0_PFLM		B0_P0_BFE
Reset	1	1	1	0	1	1	0	1	1	1	1	0	1	1	0	1

Table 185. PFCR0 field descriptions

Field	Description
0-4 BK0_APC	Bank0 Address Pipelining Control This field controls the number of cycles between Flash array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to 0b00011 by hardware reset. 00000 Accesses may be initiated on consecutive (back-to-back) cycles. 00001 Access requests require one additional hold cycle. 00010 Access requests require two additional hold cycles. ... 11110 Access requests require 30 additional hold cycles. 11111 Access requests require 31 additional hold cycles.
5-9 BK0_WWSC	Bank0 Write Wait State Control This field controls the number of wait states to be added to the Flash array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to an appropriate value by hardware reset. This field is set to 0b00011 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.

Table 185. PFCR0 field descriptions (continued)

Field	Description
10-14 BK0_RWSC	Bank0 Read Wait State Control This field controls the number of wait states to be added to the Flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. 0 MHz, < 23 MHz APC = RWSC = 0. 23 MHz, < 45 MHz APC = RWSC = 1. 45 MHz, < 68 MHz APC = RWSC = 2. 68 MHz, < 90 MHz APC = RWSC = 3. This field is set to 0b00011 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.
15-16,24 BK0_RWWC	Bank0 Read-While-Write Control This 3-bit field defines the controller response to Flash reads while the array is busy with a program (write) or erase operation. 0*xx* Terminate any attempted read while write/erase with an error response. 100 Generate a bus stall for a read while write/erase, enable the operation termination and the abort notification interrupt. 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt. 110 Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt. 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.

Table 185. PFCR0 field descriptions (continued)

Field	Description
17-18 B0_P1_BCFG	Bank0, Port 1 Page Buffer Configuration This field controls the configuration of the four page buffers in the PFlash controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the Flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. 00 All four buffers are available for any Flash access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved. 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses. This field is set to 2b11 by hardware reset.
19 B0_P1_DPFE	Bank0, Port 1 Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access.
20 B0_P1_IPFE	Bank0, Port 1 Instruction Prefetch Enable This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. 0 No prefetching is triggered by an instruction fetch read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access.
21-22 B0_P1_PFLM	Bank0, Port 1 Prefetch Limit This field controls the prefetch algorithm used by the PFlash controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is set to 2b10 by hardware reset. 00 No prefetching is performed. 01 The referenced line is prefetched on a buffer miss, that is, *prefetch on miss*. 1*x* The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, *prefetch on miss or hit*.
23 B0_P1_BFE	Bank0, Port 1 Buffer Enable This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset. 0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

Table 185. PFCR0 field descriptions (continued)

Field	Description
25-26 B0_P0_BCFG	Bank0, Port 0 Page Buffer Configuration This field controls the configuration of the four page buffers in the PFlash controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the Flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. 00 All four buffers are available for any Flash access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved. 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses. This field is set to 2b11 by hardware reset.
27 B0_P0_DPFE	Bank0, Port 0 Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access.
28 B0_P0_IPFE	Bank0, Port 0 Instruction Prefetch Enable This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. 0 No prefetching is triggered by an instruction fetch read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access.
29-30 B0_P0_PFLM	Bank0, Port 0 Prefetch Limit This field controls the prefetch algorithm used by the PFlash controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is set to 2b10 by hardware reset. 00 No prefetching is performed. 01 The referenced line is prefetched on a buffer miss, that is, *prefetch on miss*. 1*x* The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, *prefetch on miss or hit*.
31 B0_P0_BFE	Bank0, Port 0 Buffer Enable This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset. 0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

Platform Flash Configuration Register 1 (PFCR1)

The Platform Flash Configuration Register 1 (PFCR1) defines the configuration associated with Flash memory bank1. This corresponds to the data Flash. The register is described in *Figure 186* and *Table 186*.

Note: This register is not implemented on the data Flash block.

Figure 186. Platform Flash Configuration Register 1 (PFCR1)

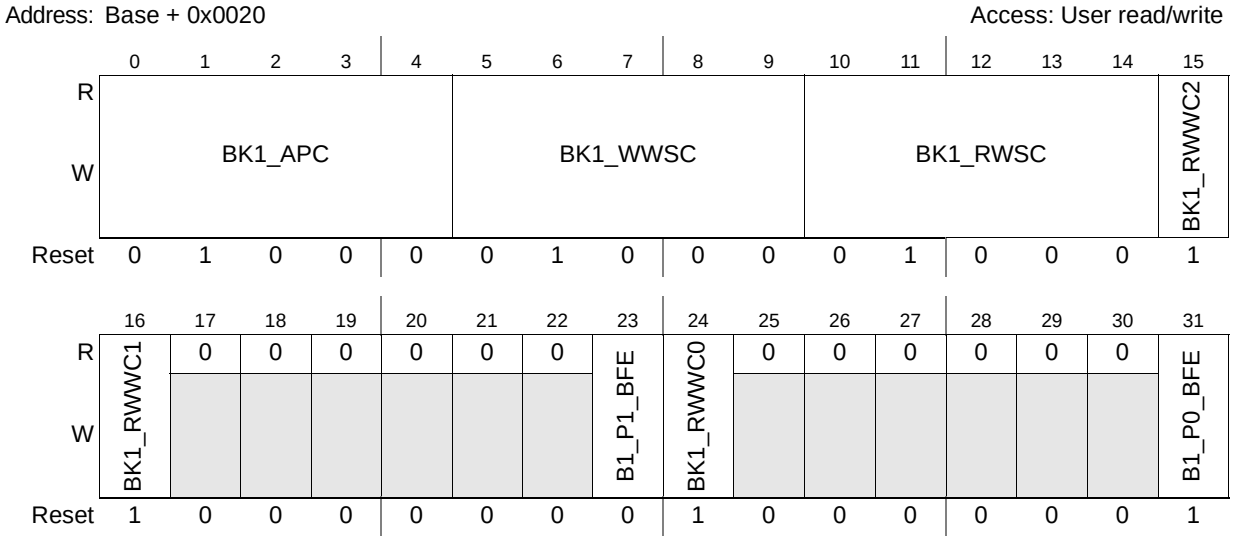


Table 186. PFCR1 field descriptions

Field	Description
0-4 BK1_APC	Bank1 Address Pipelining Control This field controls the number of cycles between Flash array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to 0b01000 by hardware reset. 00000 Accesses may be initiated on consecutive (back-to-back) cycles. 00001 Access requests require one additional hold cycle. 00010 Access requests require two additional hold cycles. ... 11110 Access requests require 30 additional hold cycles. 11111 Access requests require 31 additional hold cycles.

Table 186. PFCR1 field descriptions (continued)

Field	Description
5-9 BK1_WWSC	Bank1 Write Wait State Control This field controls the number of wait states to be added to the Flash array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to an appropriate value by hardware reset. This field is set to 0b01000 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.
10-14 BK1_RWSC	Bank1 Read Wait State Control This field controls the number of wait states to be added to the Flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. 0 MHz, < 23 MHz APC = RWSC = 0. 23 MHz, < 45 MHz APC = RWSC = 1. 45 MHz, < 68 MHz APC = RWSC = 2. 68 MHz, < 90 MHz APC = RWSC = 3. This field is set to 0b01000 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.
15-16,24 BK1_RWWC	Bank1 Read-While-Write Control This 3-bit field defines the controller response to Flash reads while the array is busy with a program (write) or erase operation. 0*xx* Terminate any attempted read while write/erase with an error response. 100 Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt. 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt. 110 Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt. 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.
17-22	Reserved, should be cleared.

Table 186. PFCR1 field descriptions (continued)

Field	Description
23 B1_P1_BFE	Bank1, Port 1 Buffer Enable This bit enables or disables read hits from the 128-bit holding register. It is also used to invalidate the contents of the holding register. This bit is cleared by hardware reset, enabling the use of the holding register. 0 The holding register is disabled from satisfying read requests. 1 The holding register is enabled to satisfy read requests on hits.
25-30	Reserved, should be cleared.
31 B1_P0_PFE	Bank1, Port 0 Buffer Enable This bit enables or disables read hits from the 128-bit holding register. It is also used to invalidate the contents of the holding register. This bit is set by hardware reset, enabling the use of the holding register. 0 The holding register is disabled from satisfying read requests. 1 The holding register is enabled to satisfy read requests on hits.

Platform Flash Access Protection Register (PFAPR)

The Platform Flash Access Protection Register (PFAPR) controls read and write accesses to the Flash based on system master number. Prefetching capabilities are defined on a per master basis. This register also defines the arbitration mode for controllers supporting two AHB ports. The register is described in *Figure 187* and *Table 187*.

The contents of the register are loaded from location 0x20_3E00 of the shadow region in the code Flash (bank0) array at reset. To temporarily change the values of any of the fields in the PFAPR, a write to the IPS-mapped register is performed. To change the values loaded into the PFAPR *at reset*, the word location at address 0x20_3E00 of the shadow region in the Flash array must be programmed using the normal sequence of operations. The reset value shown in *Table 187* reflects an erased or unprogrammed value from the shadow region.

Note: This register is not implemented on the data Flash block.

Figure 187. Platform Flash Access Protection Register (PFAPR)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	ARBM		M7 PFD	M6 PFD	M5 PFD	M4 PFD	M3 PFD	M2 PFD	M1 PFD	M0 PFD
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M7AP		M6AP		M5AP		M4AP		M3AP		M2AP		M1AP		M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 187. PFAPR field descriptions

Field	Description
0-5	Reserved, should be cleared.
6-7 ARBM	Arbitration Mode This 2-bit field controls the arbitration for PFlash controllers supporting 2 AHB ports. 00 Fixed priority arbitration with AHB p0 > p1. 01 Fixed priority arbitration with AHB p1 > p0. 1*x* Round-robin arbitration.
8-15 M*x*PFD	Master *x* Prefetch Disable (*x* = 0,1,2,...,7) These bits control whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_P*x*_DPFE, B0_P*x*_IPFE, B*x*_P*y*_BFE] bits. 0 Prefetching may be triggered by this master. 1 No prefetching may be triggered by this master.
16-31 M*x*AP	Master *x* Access Protection (*x* = 0,1,2,...,7) These fields control whether read and write accesses to the Flash are allowed based on the master number of the initiating module. 00 No accesses may be performed by this master. 01 Only read accesses may be performed by this master. 10 Only write accesses may be performed by this master. 11 Both read and write accesses may be performed by this master.

Non-Volatile Bus Interface Unit 2 register (NVBIU2)

The NVBIU2 register is a 64-bit register, the 32 most significant bits of which (bits 63–32) are “don’t care” bits that are eventually used to manage ECC codes.

Figure 188. Non-Volatile Bus Interface Unit 2 register (NVBIU2)

Address: Base + 0x0x20_3E00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BI2 31	BI2 30	BI2 29	BI2 28	BI2 27	BI2 26	BI2 25	BI2 24	BI2 23	BI2 22	BI2 21	BI2 20	BI2 19	BI2 18	BI2 17	BI2 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI2 15	BI2 14	BI2 13	BI2 12	BI2 11	BI2 10	BI2 9	BI2 8	BI2 7	BI2 6	BI2 5	BI2 4	BI2 3	BI2 2	BI2 1	BI2 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 188. BIU2 and NVBIU2 field descriptions

Field	Description
BI2[31:0] 0:31	Bus Interface unit 2 31–0 (Read/Write) The BI2[31:00] generic registers are reset based on the information stored in NVBIU2. Write access to the bits in this register can be locked.

Bus Interface Unit 3 register (BIU3)

The Bus Interface Unit 3 register provides a means for BIU specific information, or BIU configuration information to be stored.

The BIU3 register has a related Non-Volatile Bus Interface Unit 3 register located in Shadow Sector that contains the default reset value for BIU3: the NVBIU3 register is read during the reset phase of the Flash module and loaded into the BIU3.

Figure 189. Bus Interface Unit 3 register (BIU3)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BI3 31	BI3 30	BI3 29	BI3 28	BI3 27	BI3 26	BI3 25	BI3 24	BI3 23	BI3 22	BI3 21	BI3 20	BI3 19	BI3 18	BI3 17	BI3 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI3 15	BI3 14	BI3 13	BI3 12	BI3 11	BI3 10	BI3 9	BI3 8	BI3 7	BI3 6	BI3 5	BI3 4	BI3 3	BI3 2	BI3 1	BI3 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Non-Volatile Bus Interface Unit 3 register (NVBIU3)

The NVBIU3 register is a 64 bit register, the 32 most significant bits of which (bits 63–32) are “don’t care” bits that are eventually used to manage ECC codes.

Figure 190. Non-Volatile Bus Interface Unit 3 register (NVBIU3)

Address: 0x20_3E08 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BI3 31	BI3 30	BI3 29	BI3 28	BI3 27	BI3 26	BI3 25	BI3 24	BI3 23	BI3 22	BI3 21	BI3 20	BI3 19	BI3 18	BI3 17	BI3 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI3 15	BI3 14	BI3 13	BI3 12	BI3 11	BI3 10	BI3 9	BI3 8	BI3 7	BI3 6	BI3 5	BI3 4	BI3 3	BI3 2	BI3 1	BI3 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 189. BIU3 and NVBIU3 field descriptions

Field	Description
BI3[31:0] 0:31	Bus Interface unit 3 31-00 (Read/Write) The BI3[31:0] bits are reset based on the information stored in NVBIU3. The writability of the bits in this register can be locked.

User Test 0 register (UT0)

The User Test feature gives the user of the Flash module the ability to perform test features on the Flash. The User Test 0 register allows controlling the way in which the Flash content check is done.

The UT0[MRE], UT0[MRV], UT0[AIS], UT0[EIE], and DSI[7:0] bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 191. User Test 0 register (UT0)

Address: Base + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 190. UT0 field descriptions

Field	Description
UTE 0	User Test Enable This status bit indicates when User Test is enabled. All bits in UT0–2 and UMISR0–4 are locked when this bit is 0. This bit is not writeable to a 1, but may be cleared. The reset value is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. For UTE the password 0xF9F9_9999 must be written to the UT0 register.
1:7	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
DSI7-0 8:15	Data Syndrome Input 7–0 These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. The DSI7–0 bits correspond to the 8 syndrome bits on a double word. These bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 The syndrome bit is forced at 0. 1 The syndrome bit is forced at 1.

Table 190. UT0 field descriptions (continued)

Field	Description
16:24	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
25	*Reserved* (Read/Write) This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.
MRE 26	Margin Read Enable MRE enables margin reads to be done. This bit, combined with MRV, enables regular user mode reads to be replaced by margin reads. Margin reads are only active during Array Integrity Checks; Normal user reads are not affected by MRE. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Margin reads are disabled. All reads are User mode reads. 1 Margin reads are enabled.
MRV 27	Margin Read Value If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Zero's (programmed) margin reads are requested (if MRE = 1). 1 One's (erased) margin reads are requested (if MRE = 1).
EIE 28	ECC data Input Enable EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 ECC Logic Check is disabled. 1 ECC Logic Check is enabled.
AIS 29	Array Integrity Sequence AIS determines the address sequence to be used during array integrity checks or Margin Mode. The default sequence (AIS = 0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. In Margin Mode only the linear sequence (AIS = 1) is allowed, while the proprietary sequence (AIS = 0) is forbidden. 0 Array Integrity sequence is a proprietary sequence. 1 Array Integrity or Margin Mode sequence is sequential.
AIE 31	Array Integrity Enable AIE set to 1 starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (UMISR0–4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if MCR[ERS], MCR[PGM], and MCR[EHV] are all low. 0 Array Integrity Checks are disabled. 1 Array Integrity Checks are enabled.

Table 190. UT0 field descriptions (continued)

Field	Description
AID 31	Array Integrity Done AID is cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID is set to indicate that the Array Integrity Check is complete. At this time, the MISR (UMISR0–4) can be checked. 0 Array Integrity Check is on-going. 1 Array Integrity Check is done.

User Test 1 register (UT1)

The User Test 1 register allows to enable the checks on the ECC logic related to the 32 LSB of the Double Word.

The User Test 1 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 192. User Test 1 register (UT1)

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI 31	DAI 30	DAI 29	DAI 28	DAI 27	DAI 26	DAI 25	DAI 24	DAI 23	DAI 22	DAI 21	DAI 20	DAI 19	DAI 18	DAI 17	DAI 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI 15	DAI 14	DAI 13	DAI 12	DAI 11	DAI 10	DAI 9	DAI 8	DAI 7	DAI 6	DAI 5	DAI 4	DAI 3	DAI 2	DAI 1	DAI 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 191. UT1 field descriptions

Field	Description
DAI[31:0] 0:31	Data Array Input 31–0 These bits represent the input of the even word of ECC logic used in the ECC Logic Check. The DAI[31:0] bits correspond to the 32 array bits representing Word 0 within the double word. 0 The array bit is forced at 0. 1 The array bit is forced at 1.

User Test 2 register (UT2)

The User Test 2 register allows to enable the checks on the ECC logic related to the 32 MSB of the Double Word.

The User Test 2 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Note: This register is not implemented on the data Flash block.

Figure 193. User Test 2 register (UT2)

Address: Base + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI 63	DAI 62	DAI 61	DAI 60	DAI 59	DAI 58	DAI 57	DAI 56	DAI 55	DAI 54	DAI 53	DAI 52	DAI 51	DAI 50	DAI 49	DAI 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI 47	DAI 46	DAI 45	DAI 44	DAI 43	DAI 42	DAI 41	DAI 40	DAI 39	DAI 38	DAI 37	DAI 36	DAI 35	DAI 34	DAI 33	DAI 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 192. UT2 field descriptions

Field	Description
DAI[63:32] 0:31	Data Array Input [63:32] These bits represent the input of the odd word of ECC logic used in the ECC Logic Check. The DAI[63:32] bits correspond to the 32 array bits representing Word 1 within the double word. 0 The array bit is forced at 0. 1 The array bit is forced at 1.

User Multiple Input Signature Register 0 (UMISR0)

The Multiple Input Signature Register 0 (UMISR0) provides a mean to evaluate the array integrity. UMISR0 represents the bits 31:0 of the whole 144-bit word (2 double words including ECC).

UMISR0 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 194. User Multiple Input Signature Register 0 (UMISR0)

Address: Base + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 031	MS 030	MS 029	MS 028	MS 027	MS 026	MS 025	MS 024	MS 023	MS 022	MS 021	MS 020	MS 019	MS 018	MS 017	MS 016
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 015	MS 014	MS 013	MS 012	MS 011	MS 010	MS 009	MS 008	MS 007	MS 006	MS 005	MS 004	MS 003	MS 002	MS 001	MS 000
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 193. UMSIR0 field descriptions

Field	Description
MS[031:000] 0:31	Multiple input Signature 031–000 These bits represent the MISR value obtained by accumulating the bits 31:0 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR0 register.

User Multiple Input Signature Register 1 (UMISR1)

The Multiple Input Signature Register 1 (UMISR1) provides a means to evaluate the array integrity. UMISR1 represents bits 63:32 of the whole 144-bit word (2 double words including ECC).

UMISR1 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 195. User Multiple Input Signature Register 1 (UMISR1)

Address: Base + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 063	MS 062	MS 061	MS 060	MS 059	MS 058	MS 057	MS 056	MS 055	MS 054	MS 053	MS 052	MS 051	MS 050	MS 049	MS 048
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 047	MS 046	MS 045	MS 044	MS 043	MS 042	MS 041	MS 040	MS 039	MS 038	MS 037	MS 036	MS 035	MS 034	MS 033	MS 032
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 194. UMISR1 field descriptions

Field	Description
MS[063:032] 0:31	Multiple input Signature 063–032 These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR1 register.

User Multiple Input Signature Register 2 (UMISR2)

The Multiple Input Signature Register (UMISR2) provides a mean to evaluate the array integrity. UMISR2 represents the bits 95-64 of the whole 144-bit word (2 double words including ECC).

UMISR2 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Note: This register is not implemented on the data Flash block.

Figure 196. User Multiple Input Signature Register 2 (UMISR2)

Address: Base + 0x0050 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 095	MS 094	MS 093	MS 092	MS 091	MS 090	MS 089	MS 088	MS 087	MS 086	MS 085	MS 084	MS 083	MS 082	MS 081	MS 080
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 079	MS 078	MS 077	MS 076	MS 075	MS 074	MS 073	MS 072	MS 071	MS 070	MS 069	MS 068	MS 067	MS 066	MS 065	MS 064
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 195. UMISR2 field descriptions

Field	Description
MS[095:064] 0:31	Multiple input Signature 095–064 These bits represent the MISR value obtained by accumulating the bits 95:64 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR2 register.

User Multiple Input Signature Register 3 (UMISR3)

The Multiple Input Signature Register 3 (UMISR3) provides a means to evaluate the array integrity. UMISR3 represents bits 127:96 of the whole 144-bit word (2 double words including ECC).

UMISR3 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Note: This register is not implemented on the data Flash block.

Figure 197. User Multiple Input Signature Register 3 (UMISR3)

Address: Base + 0x0054 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 127	MS 126	MS 125	MS 124	MS 123	MS 122	MS 121	MS 120	MS 119	MS 118	MS 117	MS 116	MS 115	MS 114	MS 113	MS 112
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 111	MS 110	MS 109	MS 108	MS 107	MS 106	MS 105	MS 104	MS 103	MS 102	MS 101	MS 100	MS 099	MS 098	MS 097	MS 096
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 196. UMISR3 field descriptions

Field	Description
MS[127:096] 0:31	Multiple Input Signature 127–096 These bits represent the MISR value obtained accumulating bits 127:96 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR3 register.

User Multiple Input Signature Register 4 (UMISR4)

The Multiple Input Signature Register 4 (UMISR4) provides a means to evaluate the array integrity. The UMISR4 represents the ECC bits of the whole 144-bit word (2 double words including ECC). Bits 8:15 are ECC bits for the odd double word and bits 24:31 are the ECC bits for the even double word. Bits 4:5 and 20:21 of UMISR4 are the double and single ECC error detection for odd and even double words, respectively.

UMISR4 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Note: This register is not implemented on the data Flash block.

Figure 198. User Multiple Input Signature Register 4 (UMISR4)

Address: Base + 0x0058 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS
W	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS	MS
W	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 197. UMISR4 field descriptions

Field	Description
MS[159:128] 0:31	Multiple Input Signature 159:128 These bits represent the MISR value obtained accumulating: – MS[135:128]—8 ECC bits for the even double word – MS138—Single ECC error detection for even double word – MS139—Double ECC error detection for even double word – MS[151:144]—8 ECC bits for the odd double word – MS154—Single ECC error detection for odd double word – MS155—Double ECC error detection for odd double word The MS can be seeded to any value by writing the UMISR4 register.

Non-Volatile Private Censorship Password 0 register (NVPWD0)

The Non-Volatile Private Censorship Password 0 register (NVPWD0) contains the 32 LSB of the password used to validate the Censorship information contained in NVSCI0–1 registers.

Note: This register is not implemented on the data Flash block.

Figure 199. Non-Volatile private Censorship Password 0 register (NVPWD0)

Address: 0x20_3DD8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD
W	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reset	1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0

Table 198. NVPWD0 field descriptions

Field	Description
PWD[31:0] 0:31	Password 31–0 The PWD[31:0] bits represent the 32 LSB of the private censorship password.

Non-Volatile Private Censorship Password 1 register (NVPWD1)

The Non-Volatile Private Censorship Password 1 Register (NVPWD1) contains the 32 MSB of the password used to validate the Censorship information contained in NVSCI0–1 registers.

Note: This register is not implemented on the data Flash block.

Figure 200. Non-Volatile Private Censorship Password 1 register (NVPWD1)

Address: 0x20_3DDC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD
W	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
Reset	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD
W	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	1	0	1	1	1	1	1	0	1	1	1		1	1	1	1

Table 199. NVPWD1 field descriptions

Field	Description
0:31	**PWD63–32**: *PassWorD 63–32* The PWD63–32 registers represent the 32 MSB of the Private Censorship Password.

Non-Volatile System Censoring Information 0 register (NVSCI0)

The Non-Volatile System Censoring Information 0 register (NVSCI0) stores the 32 LSB of the Censorship Control Word of the device.

NVSCI0 is a non-volatile register located in Shadow sector. It is read during the reset phase of the Flash module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user. Delivery value: 0x55AA_55AA.

Note: This register is not implemented on the data Flash block.

Figure 201. Non-Volatile System Censoring Information 0 register (NVSCI0)

Address: 0x20_3DE0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC 15	SC 14	SC 13	SC 12	SC 11	SC 10	SC 9	SC 8	SC 7	SC 6	SC 5	SC 4	SC 3	SC 2	SC 1	SC 0
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CW 15	CW 14	CW 13	CW 12	CW 11	CW 10	CW 9	CW 8	CW 7	CW 6	CW 5	CW 4	CW 3	CW 2	CW 1	CW 0
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Table 200. NVSCI0 field descriptions

Field	Description
SC[15:0] 0:15	Serial Censorship control word 15–0 These bits represent the 16 LSB of the Serial Censorship Control Word (SCCW). If SC[15:0] = 0x55AA and NVSCI1 = NVSCI0, the Public Access is disabled. If SC[15:0] ≠ 0x55AA or NVSCI1 ≠ NVSCI0, the Public Access is enabled.
CW[15:0] 16:31	Censorship control Word 15–0 These bits represent the 16 LSB of the Censorship Control Word (CCW). If CW[15:0] = 0x55AA and NVSCI1 = NVSCI0, the Censored mode is disabled. If CW[15:0] ≠ 0x55AA or NVSCI1 ≠ NVSCI0, the Censored mode is enabled.

Non-Volatile System Censoring Information 1 register (NVSCI1)

The Non-Volatile System Censoring Information 1 register (NVSCI1) stores the 32 MSB of the Censorship Control Word of the device.

NVSCI1 is a non-volatile register located in Shadow sector. It is read during the reset phase of the Flash module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user. Delivery value: 0x55AA_55AA.

Note: This register is not implemented on the data Flash block.

Figure 202. Non-Volatile System Censoring Information 1 register (NVSCI1)

Address: 0x20_3DE4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC 31	SC 30	SC 29	SC 28	SC 27	SC 26	SC 25	SC 24	SC 23	SC 22	SC 21	SC 20	SC 19	SC 18	SC 17	SC 16
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CW 31	CW 30	CW 29	CW 28	CW 27	CW 26	CW 25	CW 24	CW 23	CW 22	CW 21	CW 20	CW 19	CW 18	CW 17	CW 16
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Table 201. NVSCI1 field descriptions

Field	Description
SC[32:16] 0:15	Serial Censorship control word 32–16 These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW). If SC[32:16] = 0x55AA and NVSCI1 = NVSCI0, the Public Access is disabled. If SC[32:16] ≠ 0x55AA or NVSCI1≠ NVSCI0, the Public Access is enabled.
CW[32:16] 16:31	Censorship control Word 32–16 These bits represent the 16 MSB of the Censorship Control Word (CCW). CW[32:16] = 0x55AA and NVSCI1 = NVSCI0, the Censored mode is disabled. CW[32:16]≠ 0x55AA or NVSCI1≠ NVSCI0, the Censored mode is enabled.

Non-Volatile User Options register (NVUSRO)

The Non-Volatile User Options Register (NVUSRO) contains configuration information for the user application.

NVUSRO is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes.

Note: This register is not implemented on the data Flash block.

Figure 203. Non-Volatile User Options register (NVUSRO)

Address: 0x20_3E18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WATCHDOG_EN	UO30	PAD3V5V	UO28	UO27	UO26	UO25	UO24	UO23	UO22	UO21	UO20	UO19	UO18	UO17	UO16
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	UO15	UO14	UO13	UO12	UO11	UO10	UO9	UO8	UO7	UO6	UO5	UO4	UO3	UO2	UO1	UO0
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

The Non-Volatile User Options Register (NVUSRO) contains configuration information for the user application.

NVUSRO is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes.

Table 202. NVUSRO field descriptions

Field	Description
UO[28:0] 3:31	User Options 28–0 The UO[28:0] bits are reset based on the information stored in NVUSRO.
PAD3V5V 2	PAD3V5V 0 High Voltage supply is 5.0 V. 1 High Voltage supply is 3.3 V. Default manufacturing value before Flash initialization is '1' (3.3 V), which should ensure correct minimum slope for boundary scan.
UO[30] 1	User Option 30 The UO[30] bit is reset based on the information stored in NVUSRO.
WATCHDOG_EN 0	Watchdog Enable 0 Disable after reset. 1 Enable after reset. Default manufacturing value before Flash initialization is '1' The WATCHDOG_EN bit is available only for SWT_0.

17.3.8 Code Flash programming considerations

Modify operation

All the modify operations of the Flash module are managed through the Flash user registers interface.

All the sectors of the Flash module belong to the same partition (Bank), therefore when a Modify operation is active on some sectors, no read access is possible on any other sector (Read-While-Modify is not supported).

During a Flash modify operation, any attempt to read any Flash location will output invalid data and the MCR[RWE] bit will be automatically set. This means that the Flash block is not fetchable when a modify operation is active and these commands must be executed from another memory (internal RAM or another Flash block).

If a reset occurs during a modify operation, the operation is interrupted and the block is reset to read mode. The data integrity of the Flash section where the modify operation has been terminated is not guaranteed. The interrupted Flash modify operation must be repeated.

In general, each modify operation is started through a sequence of three steps:

1. The first instruction selects the desired operation by setting its corresponding selection bit in MCR (MCR[PGM] or MCR[ERS]) or UT0 (UT0[MRE] or UT0[EIE]).
2. The second step defines the operands: the address and the data for programming or the sectors for erase or margin read.
3. The third instruction starts the modify operation by setting MCR[EHV] or UT0[AIE].

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

A summary of the available Flash modify operations are shown in *Table 203*.

Table 203. Flash modify operations

Operation	Select bit	Operands	Start bit
Double word program	MCR[PGM]	Address and data by interlock writes	MCR[EHV]
Sector erase	MCR[ERS]	LMS	MCR[EHV]
Array integrity check	None	LMS	UT0[AIE]
Margin read	UT0[MRE]	UT0[MRV] + LMS	UT0[AIE]
ECC logic check	UT0[EIE]	UT0[DSI], UT1, UT2	UT0[AIE]

Once MCR[EHV] (or UT0[AIE]) is set, the operands cannot be modified until MCR[DONE] (or UT0[AID]) is high.

In general, each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for bit MCR[DONE] (or UT0[AID]) to go high.
2. Check operation result: check bit MCR[PEG] (or compare UMISR0–4 with expected value).
3. Switch off FPEC by resetting MCR[EHV] (or UT0[AIE]).
4. Deselect current operation by clearing MCR[PGM] and MCR[ERS] (or UT0[MRE] and UT0[EIE]).

If a modify operation is on-going in one of the Flash blocks, it is forbidden to start any other modify operation on the other Flash block.

In the following sections, all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

Double word program

A Flash program sequence operates on any double word within the Flash core.

One or both words within a double word may be altered in a single program operation.

Whenever the Flash is programmed, ECC bits are also programmed (unless the selected address belongs to a sector in which the ECC has been disabled in order to allow bit manipulation). ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be of 64 bits. The programming operation should completely fill selected ECC segments within the double word.

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1.

Addresses in locked/disabled blocks cannot be programmed.

The user may program the values in any or all of 2 words of a double word with a single program sequence.

Double word-bound words have addresses that differ only in address bit 2.

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.
   a) Write the first address to be programmed with the program data.
   b) The Flash module latches address bits (22:3) at this time.
   c) The Flash module latches data written as well.
   d) This write is referred to as a program data interlock write. An interlock write may be as large as 64 bits, and as small as 32 bits (depending on the CPU bus).
3. If more than 1 word is to be programmed, write the additional address in the double word with data to be programmed. This is referred to as a program data write.
   The Flash module ignores address bits (22:3) for program data writes.
   The eventual unwritten data word defaults to 0xFFFF_FFFF.
4. Set MCR[EHV] to 1 to start the internal program sequence, or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

A program operation may be initiated with the 0-to-1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program.

The first write after a program is initiated determines the page address to be programmed. This first write is referred to as an interlock write. The interlock write determines whether the shadow, test, or normal array space will be programmed by causing MCR[PEAS] to be set/cleared.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

After the interlock write, additional writes only affect the data to be programmed at the word location determined by address bit 2. Unwritten locations default to a data value of

0xFFFF_FFFF. If multiple writes are done to the same location, the data for the last write is used in programming.

While MCR[DONE] is low and MCR[EHV] is high, the user may clear MCR[EHV], resulting in a program terminate. A program termination forces the module to step 8 of the program sequence.

A terminated program results in MCR[PEG] being cleared, indicating a failed operation. MCR[DONE] must be checked to know when the terminating command has completed.

The data space being operated on before the termination will contain indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

Example 1 Double word program of data 0x55AA_55AA at address 0x00_AAA8 and data 0xAA55_AA55 at address 0x00_AAAC

```
MCR        = 0x00000010;        /* Set PGM in MCR: Select Operation */
(0x00AAA8) = 0x55AA55AA;        /* Latch Address and 32 LSB data */
(0x00AAAC) = 0xAA55AA55;        /* Latch 32 MSB data */
MCR        = 0x00000011;        /* Set EHV in MCR: Operation Start */
do                              /* Loop to wait for DONE=1 */
{ tmp      = MCR;               /* Read MCR */
} while ( !(tmp & 0x00000400) );
status     = MCR & 0x00000200;  /* Check PEG flag */
MCR        = 0x00000010;        /* Reset EHV in MCR: Operation End */
MCR              = 0x00000000;                    /* Reset PGM in MCR:
Deselect Operation */
```

Sector erase

Erase changes the value stored in all bits of the selected block(s) to logic 1.

An erase sequence operates on any combination of blocks (sectors) in the low, mid or high address space, or the shadow block (if available). The test block cannot be erased.

The erase sequence is fully automated within the Flash. The user only needs to select the blocks to be erased and initiate the erase sequence.

Locked/disabled blocks cannot be erased.

If multiple blocks are selected for erase during an erase sequence, no specific operation order must be assumed.

The Erase operation consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to 1.
2. Select the block(s) to be erased by writing 1s to the LMS register. If the shadow block is to be erased, this step may be skipped, and LMS is ignored.

Note: Lock and Select are independent. If a block is selected and locked, no erase will occur.

3. Write to any address in Flash. This is referred to as an erase interlock write.
4. Set MCR[EHV] to start the internal erase sequence, or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Clear the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase operation.

After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to 1. Data words written during erase sequence interlock writes are ignored.

The user may terminate the erase sequence by clearing MCR[ERS] before setting MCR[EHV].

An erase operation may be terminated by clearing MCR[EHV], assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low.

An erase termination forces the Module to step 8 of the erase sequence.

A terminated erase results in MCR[PEG] being cleared, indicating a failed operation. MCR[DONE] must be checked to know when the terminating command has completed.

The block(s) being operated on before the termination contain indeterminate data. This may be recovered by executing an erase on the affected blocks.

The user may not terminate an erase sequence while in erase suspend.

Example 2 Erase of sectors B0F1 and B0F2

```
MCR        = 0x00000004;   /* Set ERS in MCR: Select Operation */
LMS        = 0x00000006;   /* Set LSL2-1 in LMS: Select Sectors to erase */
(0x000000) = 0xFFFFFFFF;   /* Latch a Flash Address with any data */
MCR        = 0x00000005;   /* Set EHV in MCR: Operation Start */
do                         /* Loop to wait for DONE=1 */
{ tmp      = MCR;          /* Read MCR */
} while ( !(tmp & 0x00000400) );
status     = MCR & 0x00000200;/* Check PEG flag */
MCR        = 0x00000004;   /* Reset EHV in MCR: Operation End */
MCR              = 0x00000000;              /* Reset ERS in MCR: Deselect
Operation */
```

The erase sequence may be suspended to allow read access to the Flash core.

It is not possible to program or to erase during an erase suspend.

During erase suspend, all reads to blocks targeted for erase return indeterminate data.

An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from 0 to 1. MCR[ESUS] can be set to 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0-to-1 transition of MCR[ESUS] causes the module to start the sequence that places it in erase suspend.

The user must wait until MCR[DONE] = 1 before the Module is suspended and further actions are attempted. MCR[DONE] will go high no more than t_{ESUS} after MCR[ESUS] is set to 1.

Once suspended, the array may be read. Flash core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

Example 3 Sector Erase Suspend

```
MCR        = 0x00000007;        /* Set ESUS in MCR: Erase Suspend */
do                              /* Loop to wait for DONE=1 */
{ tmp      = MCR;               /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Notice that there is no need to clear MCR[EHV] and MCR[ERS] in order to perform reads during erase suspend.

The Erase sequence is resumed by writing a logic 0 to MCR[ESUS].

MCR[EHV] must be set to 1 before MCR[ESUS] can be cleared to resume the operation.

The Module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

Example 4 Sector Erase Resume

```
MCR                = 0x00000005;                        /* Reset ESUS in MCR:
Erase Resume */
```

User Test mode

User Test mode is a customer-accessible mode that can be used to perform specific tests to check the integrity of the Flash module. Three kinds of test can be performed:

- Array integrity self-check
- Margin mode read
- ECC logic check

The User Test mode is equivalent to a Modify operation. Read accesses attempted by the user during User Test mode generate a Read-While-Write Error (the MCR[RWE] bit is set).

User Test operations are not allowed on the Test and Shadow blocks.

Array integrity self check

Array integrity is checked using a predefined address sequence (proprietary), and this operation is executed on selected and unlocked blocks. Once the operation is completed, the results of the reads can be checked by reading the MISR value (stored in UMISR0–4), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32-bit register.

The 128-bit data, the 16-bit ECC data, and the single and double ECC errors of the two double words are therefore captured by the MISR through five different read accesses at the same location.

The whole check is done through five complete scans of the memory address space:

1. The first pass scans only bits 31:0 of each page.
2. The second pass scans only bits 63:32 of each page.
3. The third pass scans only bits 95:64 of each page.
4. The fourth pass scans only bits 127:96 of each page.
5. The fifth pass scans only the ECC bits (8 + 8) and the single and double ECC errors (2 + 2) of both double words of each page.

The 128-bit data and the 16-bit ECC data are sampled before the eventual ECC correction, while the single and double error flags are sampled after the ECC evaluation.

Only data from existing and unlocked locations are captured by the MISR.

The MISR can be seeded to any value by writing the UMISR0–4 registers.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the LMS register.

Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
5. Wait until the UT0[AID] bit is set.
6. Compare the contents of the UMISR0–4 registers with the expected results.
7. Write a logic 0 to the UT0[AIE] bit.
8. If more blocks are to be checked, return to step 2.

It is recommended to leave UT0[AIS] at 0 and use the proprietary address sequence that checks the read path more fully, although this sequence takes more time.

Note: During the execution of the Array Integrity Check operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

While UT0[AID] is low and UT0[AIE] is high, the user may clear UT0[AIE], resulting in an Array Integrity Check termination.

UT0[AID] must be checked to know when the terminating command has completed.

Example 5 Array Integrity Check of sectors B0F1 and B0F2

```
UT0        = 0xF9F99999;   /* Set UTE in UT0: Enable User Test */
LMS        = 0x00000006;   /* Set LSL2-1 in LMS: Select Sectors */
UT0        = 0x80000002;   /* Set AIE in UT0: Operation Start */
do                         /* Loop to wait for AID=1 */
{ tmp      = UT0;          /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0      = UMISR0;       /* Read UMISR0 content*/
data1      = UMISR1;       /* Read UMISR1 content*/
data2      = UMISR2;       /* Read UMISR2 content*/
data3      = UMISR3;       /* Read UMISR3 content*/
data4      = UMISR4;       /* Read UMISR4 content*/
UT0              = 0x00000000;             /* Reset UTE and AIE in UT0:
Operation End */
```

Margin read

The Margin read procedure (either Margin 0 or Margin 1) can be run on unlocked blocks in order to unbalance the Sense Amplifiers with respect to standard read conditions so that all the read accesses reduce the margin vs. 0 (UT0[MRV] = 0) or vs. 1 (UT0[MRV] = 1). Locked sectors are ignored by MISR calculation and ECC flagging.

The results of the margin reads can be checked by comparing the checksum value in UMISR[0:4].

Since Margin reads are done at voltages that differ than the normal read voltage, the lifetime expectancy of the Flash macrocell is impacted by the execution of Margin reads.

Repeated Margin reads will result in degradation of the Flash array, and will shorten the expected lifetime experienced at normal read levels.

For these reasons, Margin reads are allowed only at the factory. Margin reads are forbidden for use by user applications.

In any case the charge losses detected through the Margin mode cannot be considered failures of the device and no failure analysis will be opened on them.

The Margin Read Setup operation consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the LMS register.

Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set UT0[AIS] bit for a sequential addressing only.
4. Change the value in the UT0[MRE] bit from 0 to 1.
5. Select the Margin level: UT0[MRV] = 0 for 0s margin, UT0[MRV] = 1 for 1s margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR[0:4] content with the expected result.
9. Write a logic 0 to the UT0[AIE], UT0[MRE] and UT0[MRV] bits.
10. If more blocks are to be checked, return to step 2.

It is mandatory to leave UT0[AIS] at 1 and use the linear address sequence (that also takes less time).

During the execution of the Margin Mode operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

The read accesses will be done with the addition of a proper number of wait states to guarantee the correctness of the result.

While UT0[AID] is low and UT0[AIE] is high, the user may clear AIE, resulting in a Array Integrity Check termination.

UT0[AID] must be checked to know when the terminating command has completed.

Example 6 Margin Read Check versus 1s

```
UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors */
UT0 = 0x80000004; /* Set AIS in UT0: Select Operation */
UT0 = 0x80000024; /* Set MRE in UT0: Select Operation */
UT0 = 0x80000034; /* Set MRV in UT0: Select Margin versus 1's */
UT0 = 0x80000036; /* Set AIE in UT0: Operation Start */
do /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content*/
data1 = UMISR1; /* Read UMISR1 content*/
data2 = UMISR2; /* Read UMISR2 content*/
data3 = UMISR3; /* Read UMISR3 content*/
data4 = UMISR4; /* Read UMISR4 content*/
UT0 = 0x80000034; /* Reset AIE in UT0: Operation End */
UT0 = 0x00000000; /* Reset UTE, MRE, MRV, AIS in UT0: Deselect Op. */
```

ECC logic check

ECC logic can be checked by forcing the input of ECC logic: The 64 bits of data and the 8 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the whole page (2 double words).

The results of the ECC Logic Check can be verified by reading the MISR value.

The ECC Logic Check operation consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Write the double word input value to UT1[DAI31–0] and UT2[DAI[63–32].
3. Write the Syndrome Input value to UT0[DSI7–0].
4. Select the ECC Logic Check: write a logic 1 to the UT0[EIE] bit.
5. Write a logic 1 to the UT0[AIE] bit to start the ECC Logic Check.
6. Wait until the UT0[AID] bit goes high.
7. Compare the contents of the UMISR0–4 registers with the expected results.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] = 0, the UMISR0–4 and UT1–2 registers and the UT0[MRE], UT0[MRV], UT0[EIE], UT0[AIS], and UT0[DSI7–0] bits are not accessible. Reads return indeterminate data; writes have no effect.

Example 7ECC logic check

```
UT0        = 0xF9F99999;   /* Set UTE in UT0: Enable User Test */
UT1        = 0x55555555;   /* Set DAI31-0 in UT1: Even Word Input Data */
UT2        = 0xAAAAAAAA;   /* Set DAI63-32 in UT2: Odd Word Input Data */
UT0        = 0x80FF0000;   /* Set DSI7-0 in UT0: Syndrome Input Data */
UT0        = 0x80FF0008;   /* Set EIE in UT0: Select ECC Logic Check */
UT0        = 0x80FF000A;   /* Set AIE in UT0: Operation Start */
do                         /* Loop to wait for AID=1 */
{ tmp      = UT0;          /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0      = UMISR0;       /* Read UMISR0 content (expected 0x55555555) */
data1      = UMISR1;       /* Read UMISR1 content (expected 0xAAAAAAAA) */
data2      = UMISR2;       /* Read UMISR2 content (expected 0x55555555) */
data3      = UMISR3;       /* Read UMISR3 content (expected 0xAAAAAAAA) */
data4      = UMISR4;       /* Read UMISR4 content (expected 0x00FF00FF) */
UT0              = 0x00000000;              /* Reset UTE, AIE and EIE in
UT0: Operation End */
```

Error Correction Code (ECC)

The Flash macrocell provides a method to improve the reliability of the data stored in Flash: the usage of an Error Correction Code. The word size is fixed at 64 bits.

Each double word of 64 bits has an associated 8 ECC bits that are programmed in such a way to guarantee a Single Error Correction and a Double Error Detection (SEC-DED).

ECC circuitry provides correction of single bit faults and is used to achieve automotive reliability targets. Some units will experience single bit corrections throughout the life of the product with no impact on product reliability.

ECC algorithms

The Flash macrocell supports the ECC algorithm “All 1s No Error”.

All 1s No Error

The All 1s No Error algorithm detects as valid any double word read on a just erased sector (all the 72 bits are 1s).

This option allows performing a Blank Check after a Sector Erase operation.

EEPROM emulation

The chosen ECC algorithm allows some bit manipulations so that a Double Word can be rewritten several times without needing an erase of the sector. This allows to use a Double Word to store flags useful for the EEPROM emulation.

As an example the chosen ECC algorithm allows to start from an All ‘1’s Double Word value and rewrite whichever of its four 16-bit Half-Words to an All ‘0’s content by keeping the same ECC value.

Table 204 shows a set of Double Words sharing the same ECC value.

Table 204. Bits manipulation: double words with the same ECC value

Double word	ECC value – All 1s No Error
0xFFFF_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_FFFF_0000	0xFF
0xFFFF_FFFF_0000_FFFF	0xFF
0xFFFF_0000_FFFF_FFFF	0xFF
0x0000_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_0000_0000	0xFF
0xFFFF_0000_FFFF_0000	0xFF
0x0000_FFFF_FFFF_0000	0xFF
0xFFFF_0000_0000_FFFF	0xFF
0x0000_FFFF_0000_FFFF	0xFF
0x0000_0000_FFFF_FFFF	0xFF
0xFFFF_0000_0000_0000	0xFF
0x0000_FFFF_0000_0000	0xFF
0x0000_0000_0000_0000	0xFF

When some Flash sectors are used to perform an EEPROM emulation, it is recomended for safety reasons to reserve at least three sectors for this purpose.

Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in Flash sectors, and Censored mode to avoid piracy.

Modify protection

The Flash modify protection information is stored in non-volatile Flash cells located in the TestFlash. This information is read once during the Flash initialization phase following the exit from reset and they are stored in Volatile registers that act as actuators.

The reset state of all the volatile modify protection registers is the protected state.

All the non-volatile modify protection registers can be programmed through a normal double word program operation at the related locations in TestFlash.

The non-volatile modify protection registers cannot be erased.

- The non-volatile modify protection registers are physically located in TestFlash. Their bits can be programmed to 0 only once, after which they cannot be restored to 1.
- The volatile modify protection registers are read/write registers containing bits that can be written at 0 or 1 by the user application.

A software mechanism is provided to independently lock/unlock each Low, Mid, and High Address Space block against program and erase.

Software locking is done through the LML (Low/Mid Address Space Block Lock Register) register.

An alternate means to enable software locking for blocks of Low Address Space only is through the SLL (Secondary Low/Mid Address Space Block Lock Register).

All these registers have a non-volatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash non-volatile image is at all 1s, meaning all sectors are locked.

By programming the non-volatile locations in TestFlash, the selected sectors can be unlocked.

Because the TestFlash is one-time programmable (that is, not erasable), once the sectors have been unlocked, they cannot be locked again.

Of course, on the contrary, all the volatile registers can be written at 0 or 1 at any time, therefore the user application can lock and unlock sectors when desired.

Censored mode

The Censored mode information is stored in non-volatile Flash cells located in the Shadow Sector. This information is read once during the Flash initialization phase following the exit from Reset and they are stored in Volatile registers that act as actuators.

The reset state of all the volatile censored mode registers is the protected state.

All the non-volatile censored mode registers can be programmed through a normal double word program operation at the related locations in the Shadow Sector.

The non-volatile censored mode registers can be erased by erasing the Shadow Sector.

- The non-volatile censored mode registers are physically located in the Shadow Sector their bits can be programmed to 0 and eventually restored to 1 by erasing the Shadow Sector.
- The volatile censored mode registers are registers not accessible by the user application.

The Flash block provides two levels of protection against piracy:

- If bits NVSCI0[CW[15:0]] are programmed to 0x55AA and NVSC1 = NVSCI0, Censored mode is disabled. All other possible values enable Censored mode.
- If bits NVSCI0[SC[15:0]] are programmed to 0x55AA and NVSC1 = NVSCI0, Public Access is disabled. All other possible values enable Public Access.

The parts are delivered to the user with Censored mode and public access disabled.

The chosen Flash ECC algorithm allows to modify the censorship status without erasing the Shadow sector, as shown in *Table 205*.

Table 205. Bits manipulation: censorship management

Censored mode	Public access	NVSCI0	NVSCI1
Enabled	Enabled	0xFFFF_FFFF	0xFFFF_FFFF
Disabled	Enabled	0xFFFF_55AA	0xFFFF_55AA
Enabled	Disabled	0x55AA_FFFF	0x55AA_FFFF
Disabled	Disabled	0x55AA_55AA	0x55AA_55AA
Enabled	Disabled	0x55AA_0000	0x55AA_0000
Disabled	Enabled	0x0000_55AA	0x0000_55AA
Enabled	Enabled	0x0000_0000	0x0000_0000

18 Memory Protection Unit (MPU)

18.1 Introduction

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in a device. Using region descriptors which define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. 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 set of program-visible region descriptors which are used to monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

18.2 Block diagram

A simplified block diagram of the MPU module is shown in *Figure 204*. The hardware's two-dimensional connection matrix is clearly visible with the basic access evaluation "macro" shown as the replicated submodule block. The AHB[(h)] bus slave ports (s{0,1,2,3}_h*) are shown on the left side of the diagram, the region descriptor registers in the middle and the IPS bus interface (ips_*) on the right side. The evaluation macro contains the two magnitude comparators connected to the start and end address registers from each region descriptor (rgd*n*) as well as the combinational logic blocks to determine the region hit and the access protection error. For information on the details of the access evaluation macro, see *Section 18.7.1, Access evaluation macro*.

h. ARM's Advanced High-performance Bus

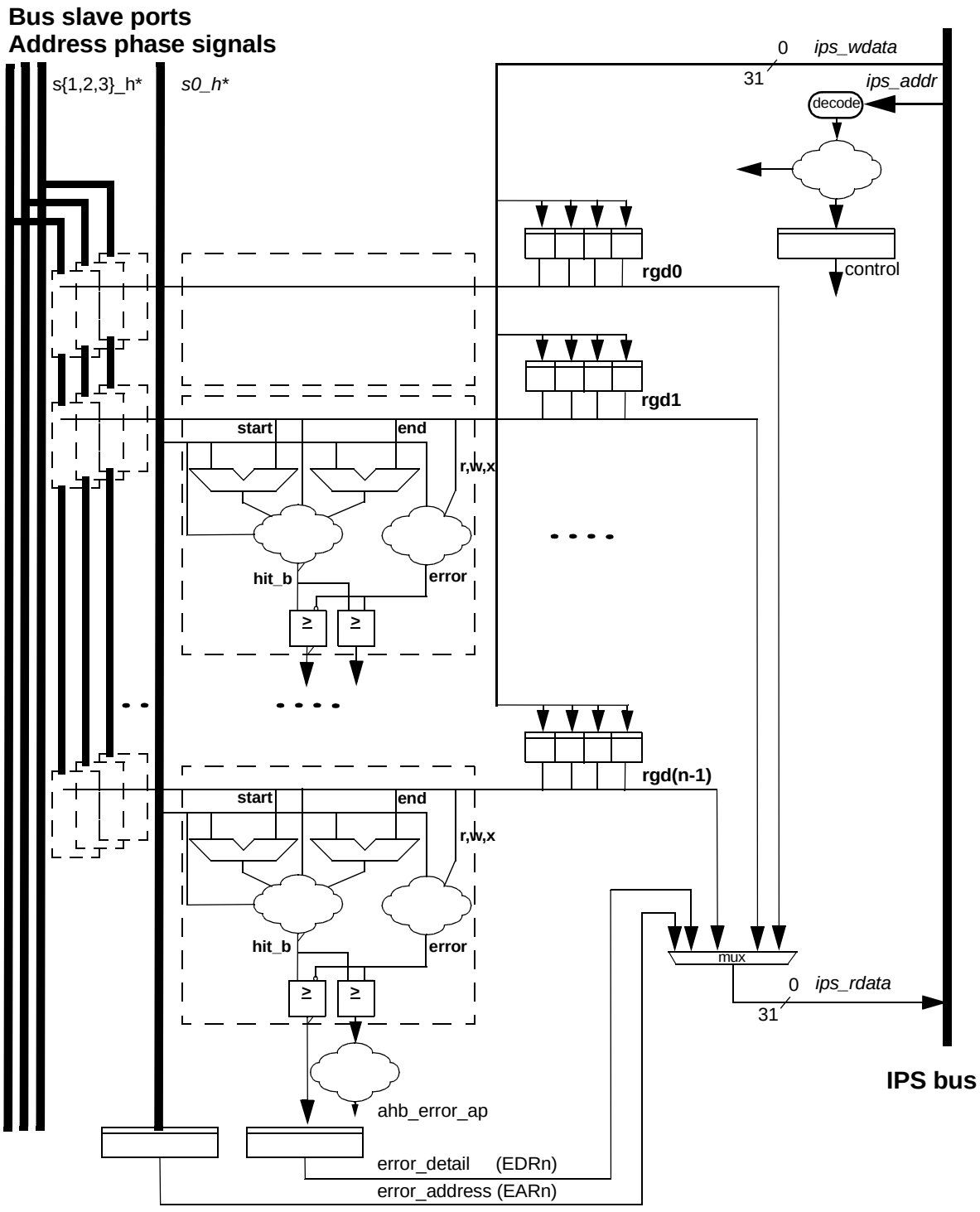


Figure 204. MPU block diagram

18.3 Features

The MPU implements a two-dimensional hardware array of memory region descriptors and the crossbar slave AHB ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for 16 memory region descriptors
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - 4 bus masters that support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses.
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter only the access rights of a descriptor
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software. See *Section 18.7.2, Putting it all together and AHB error terminations*, for details and *Section 18.9, Application information*, for an example.
- Support for 3 AHB slave port connections
  - MPU hardware continuously monitors every AHB slave port access using the pre-programmed memory region descriptors
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in *all* memory regions where it does hit. In the event of an access error, the AHB reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device. This results in a Machine Check Exception.
  - 64-bit error registers, one for each AHB slave port, capture the last faulting address, attributes and “detail” information
- Global MPU enable/disable control bit provides a mechanism to easily load region descriptors during system startup or allow complete access rights during debug with the module disabled

On this device, the two instantiations of the MPU are referred to as MPU_0 (attached to the slave side of XBAR_0) and MPU_1 (attached to the slave side of XBAR_1). Both instantiations are identical and do not differ in LS Mode or DP Mode.

The MPU port allocation is shown in *Table 206*.

Table 206. MPU port allocation

AXBS slave port	MPU_0	MPU_1
Flash memory	Port 0	Port 0
SRAM	Port 1	Port 1
Peripheral bridge	Port 2	Port 2

18.4 Modes of operation

The MPU module does not support any special modes of operation. As a memory-mapped device located on the platform’s high-speed system bus, it responds based strictly on the

memory addresses of the connected system buses. The IPS bus is used to access the MPU's programming model and the memory protection functions are evaluated on a reference-by-reference basis using the addresses from the AHB system bus port(s).

Power dissipation is minimized when the MPU's global enable/disable bit is cleared (MPU_CESR[VLD] = 0).

18.5 External signal description

The MPU module does not include any external interface.

18.6 Memory map and register definition

The MPU module provides an IPS programming model mapped to an SPP-standard on-platform 16 KB space. The programming model is partitioned into three groups:

- Control/status registers
- The data structure containing the region descriptors
- The alternate view of the region descriptor access control values

The programming model can only be referenced using 32-bit (word) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an IPS error termination.

The MPU programming model map is shown in *Table 207*.

Table 207. MPU memory map

Offset address 0xFFF1_0000 (MPU_0) 0x8FF1_0000 (MPU_1)	Register name	Register description	Size (bits)	Location
0x0000	MPU_CESR	MPU Control/Error Status Register	32	*on page 448*
0x0004–0x000F	Reserved			
0x0010	MPU_EAR0	MPU Error Address Register, Slave Port 0	32	*on page 448*
0x0014	MPU_EDR0	MPU Error Detail Register, Slave Port 0	32	*on page 449*
0x0018	MPU_EAR1	MPU Error Address Register, Slave Port 1	32	*on page 448*
0x001C	MPU_EDR1	MPU Error Detail Register, Slave Port 1	32	*on page 449*
0x0020	MPU_EAR2	MPU Error Address Register, Slave Port 2	32	*on page 448*
0x0024	MPU_EDR2	MPU Error Detail Register, Slave Port 2	32	*on page 449*
0x0028–0x03FF	Reserved			
0x0400	MPU_RGD0	MPU Region Descriptor 0	128	*on page 450*
0x0410	MPU_RGD1	MPU Region Descriptor 1	128	*on page 450*
0x0420	MPU_RGD2	MPU Region Descriptor 2	128	*on page 450*
0x0430	MPU_RGD3	MPU Region Descriptor 3	128	*on page 450*

Table 207. MPU memory map (continued)

Offset address 0xFFF1_0000 (MPU_0) 0x8FF1_0000 (MPU_1)	Register name	Register description	Size (bits)	Location
0x0440	MPU_RGD4	MPU Region Descriptor 4	128	on page 450
0x0450	MPU_RGD5	MPU Region Descriptor 5	128	on page 450
0x0460	MPU_RGD6	MPU Region Descriptor 6	128	on page 450
0x0470	MPU_RGD7	MPU Region Descriptor 7	128	on page 450
0x0480	MPU_RGD8	MPU Region Descriptor 8	128	on page 450
0x0490	MPU_RGD9	MPU Region Descriptor 9	128	on page 450
0x04A0	MPU_RGD10	MPU Region Descriptor 10	128	on page 450
0x04B0	MPU_RGD11	MPU Region Descriptor 11	128	on page 450
0x04C0	MPU_RGD12	MPU Region Descriptor 12	128	on page 450
0x04D0	MPU_RGD13	MPU Region Descriptor 13	128	on page 450
0x04E0	MPU_RGD14	MPU Region Descriptor 14	128	on page 450
0x04F0	MPU_RGD15	MPU Region Descriptor 15	128	on page 450
0x0500-0x07FF	Reserved			
0x0800	MPU_RGDAAC0	MPU RGD Alternate Access Control 0	32	on page 454
0x0804	MPU_RGDAAC1	MPU RGD Alternate Access Control 1	32	on page 454
0x0808	MPU_RGDAAC2	MPU RGD Alternate Access Control 2	32	on page 454
0x080C	MPU_RGDAAC3	MPU RGD Alternate Access Control 3	32	on page 454
0x0810	MPU_RGDAAC4	MPU RGD Alternate Access Control 4	32	on page 454
0x0814	MPU_RGDAAC5	MPU RGD Alternate Access Control 5	32	on page 454
0x0818	MPU_RGDAAC6	MPU RGD Alternate Access Control 6	32	on page 454
0x081C	MPU_RGDAAC7	MPU RGD Alternate Access Control 7	32	on page 454
0x0820	MPU_RGDAAC8	MPU RGD Alternate Access Control 8	32	on page 454
0x0824	MPU_RGDAAC9	MPU RGD Alternate Access Control 9	32	on page 454
0x0828	MPU_RGDAAC10	MPU RGD Alternate Access Control 10	32	on page 454
0x082C	MPU_RGDAAC11	MPU RGD Alternate Access Control 11	32	on page 454
0x0830	MPU_RGDAAC12	MPU RGD Alternate Access Control 12	32	on page 454
0x0834	MPU_RGDAAC13	MPU RGD Alternate Access Control 13	32	on page 454
0x0838	MPU_RGDAAC14	MPU RGD Alternate Access Control 14	32	on page 454
0x083C	MPU_RGDAAC15	MPU RGD Alternate Access Control 15	32	on page 454
0x0840-0x3FFF	Reserved			

18.6.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset MPU_Base + 0x000 Access: Read/Partial Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SPERR			0	0	0	0	0	1	0	0	0	HRL				NSP				NRGD				0	0	0	0	0	0	0	VLD
W	w1c																															
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0

Figure 205. MPU Control/Error Status Register (MPU_CESR)

Table 208. MPU_CESR field descriptions

Field	Description
SPERR	Slave Port n Error, where the slave port number matches the bit number. Each bit in this field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EARn and MPU_EDRn registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written as a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A “find first one” instruction (or equivalent) can be used to detect the presence of a captured error. 0 The corresponding MPU_EARn/MPU_EDRn registers do not contain a captured error. 1 The corresponding MPU_EARn/MPU_EDRn registers do contain a captured error.
HRL	Hardware Revision Level. This 4-bit read-only field specifies the MPU’s hardware and definition revision level. It can be read by software to determine the functional definition of the module.
NSP	Number of Slave Ports. This 4-bit read-only field specifies the number of slave ports connected to the MPU. For this device, NSP = 3.
NRGD	Number of Region Descriptors. This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0b00 8 region descriptors 0b01 12 region descriptors 0b10 16 region descriptors (correct value for this device)
VLD	Valid. This bit provides a global enable/disable for the MPU. 0 The MPU is disabled. 1 The MPU is enabled. While the MPU is disabled, all accesses from all bus masters are allowed.

18.6.2 MPU Error Address Register, Slave Port n (MPU_EARn)

When the MPU detects an access error on slave port n, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDRn register at the same time. This register and the corresponding MPU_EDRn register contain the most recent access error; there are no hardware interlocks with the MPU_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

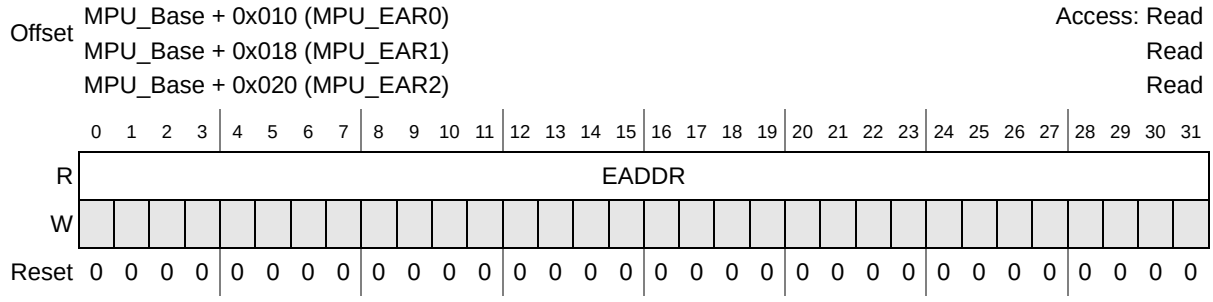


Figure 206. MPU Error Address Register, Slave Port n (MPU_EARn)

Table 209. MPU_EARn field descriptions

Field	Description
0–31 EADDR	Error Address. This read-only field is the reference address from slave port n that generated the access error.

18.6.3 MPU Error Detail Register, Slave Port n (MPU_EDRn)

When the MPU detects an access error on slave port n, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU_EARn register at the same time. Note this register and the corresponding MPU_EARn register contain the most recent access error; there are no hardware interlocks with the MPU_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

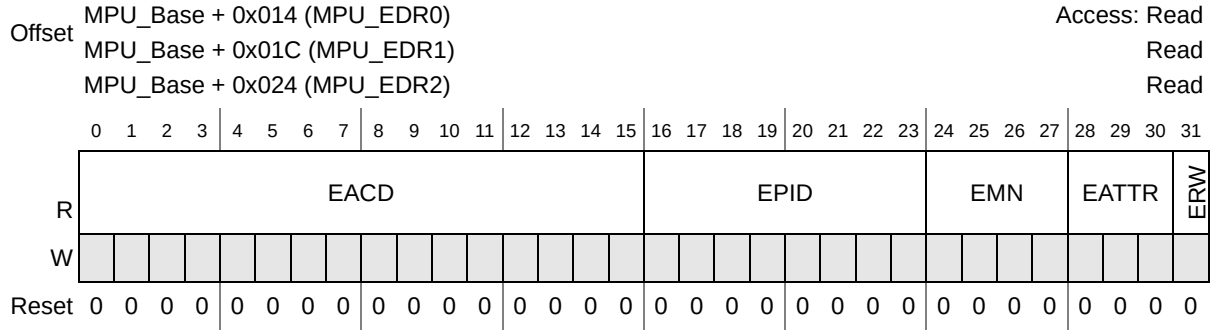


Figure 207. MPU Error Detail Register, Slave Port n (MPU_EDRn)

Table 210. MPU_EDRn field descriptions

Field	Description
0–15 EACD	Error Access Control Detail. This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit where the error occurred. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDRn register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits. If only a single EACD bit is set, then the protection error was caused by a single non-overlapping region descriptor. If two or more EACD bits are set, then the protection error was caused in an overlapping set of region descriptors.
16–23 EPID	Error Process Identification. This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven only by processor cores; for other bus masters, this field is cleared.
24–27 EMN	Error Master Number. This 4-bit read-only field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
28–30 EATTR	Error Attributes. This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 0b00 User mode, instruction access 0b01 User mode, data access 0b10 Supervisor mode, instruction access 0b11 Supervisor mode, data access All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
31 ERW	Error Read/Write. This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

18.6.4 MPU Region Descriptor n (MPU_RGDn)

Each 128-bit (four 32-bit word) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is the very essence of the operation of the Memory Protection Unit.

The region descriptors are organized sequentially in the MPU's programming model and each of the four 32-bit words are detailed in the subsequent sections.

MPU Region Descriptor n, Word 0 (MPU_RGDn.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor's valid bit (see *Section , MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3),* for more information).

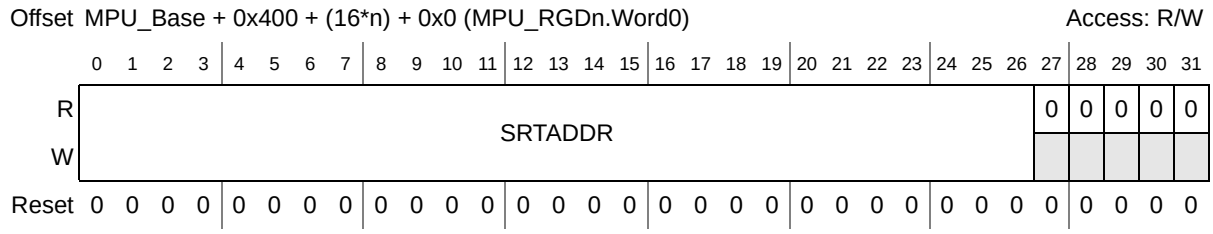


Figure 208. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0)

Table 211. MPU_RGDn.Word0 field descriptions

Field	Description
0–26 SRTADDR	Start Address. This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

MPU Region Descriptor n, Word 1 (MPU_RGDn.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor's valid bit (see *Section , MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3),* for more information).

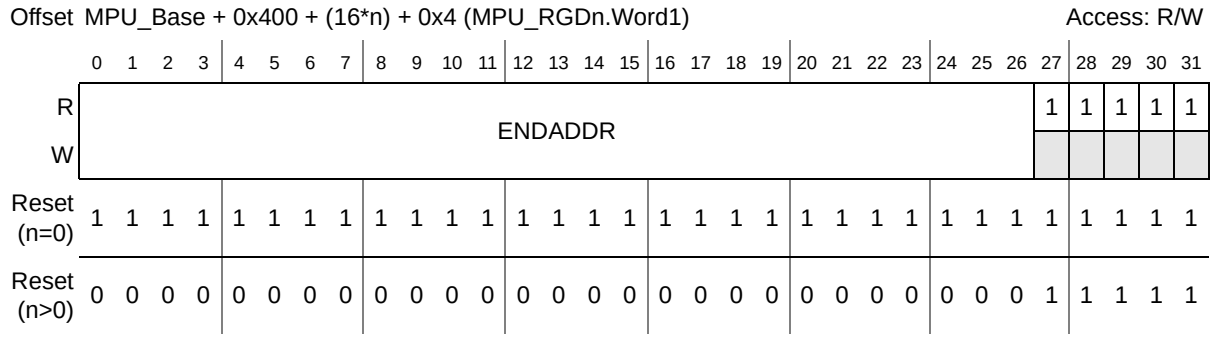


Figure 209. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1)

Table 212. MPU_RGDn.Word1 field descriptions

Field	Description
0–26 ENDADDR	End Address. This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR >= SRTADDR; it is software's responsibility to properly load these region descriptor fields.

MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. Bus masters 0-3 have a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. For these fields, the bus master number refers to the logical master number as specified in *Section 14.1.4, Logical master IDs.* (Nexus controllers have IDs 8 and 9, but only the last 3 bits of the ID are used for this purposes, so they share privileges with cores 0 and 1.)

For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

Writes to this word clear the region descriptor's valid bit (see *Section , MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3),* for more information). Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (Alternate Access Control n) as stores to these locations do not affect the descriptor's valid bit.

Offset MPU_Base + 0x400 + (16*n) + 0x8 (MPU_RGDn.Word2) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	M3PE	M3SM			M3UM				M2PE	M2SM			M2UM				M1PE	M1SM			M1UM				M0PE	M0SM			M0UM			
W													r	w	x						r	w	x						r	w	x						r	w	x	
Reset (n=0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 210. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2)

Table 213. MPU_RGDn.Word2 field descriptions

Field	Description
M3PE	Bus master 3 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M3SM	Bus master 3 supervisor mode access control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M3UM for user mode
M3UM	Bus master 3 user mode access control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

Table 213. MPU_RGDn.Word2 field descriptions (continued)

Field	Description
M2PE	Bus master 2 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M2SM	Bus master 2 supervisor mode access control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M2UM for user mode
M2UM	Bus master 2 user mode access control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus master 1 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus master 1 supervisor mode access control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M1UM for user mode
M1UM	Bus master 1 user mode access control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M0UM for user mode
M0UM	Bus master 0 user mode access control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Since the region descriptor is a 4-word entity, there are potential coherency issues as this structure is being updated since multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGDn.Word0, then MPU_RGDn.Word1,... and finally MPU_RGDn.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGDn.Word2) as different tasks execute, an alternate programming view of MPU_RGDn.Word2 is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor's valid bit.

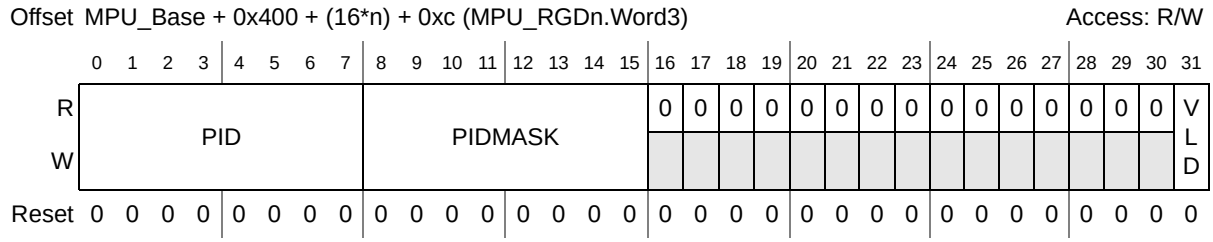


Figure 211. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3)

Table 214. MPU_RGDn.Word3 field descriptions

Field	Description
0–7 PID	Process Identifier. This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set.
8–15 PIDMASK	Process Identifier Mask. This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, then the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see *Section , Access evaluation – hit determination*.
31 VLD	Valid. This bit signals the region descriptor is valid. Any write to MPU_RGDn.Word{0,1,2} clears this bit, while a write to MPU_RGDn.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid 1 Region descriptor is valid

18.6.5 MPU Region Descriptor Alternate Access Control n (MPU_RGDAACn)

As noted in *Section , MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2)*, it is expected that since system software may adjust only the access controls within a region descriptor (MPU_RGDn.Word2) as different tasks execute, *an alternate programming view of this 32-bit entity is desired*. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor's valid bit.

The memory address therefore provides an alternate location for updating MPU_RGDn.Word2.

Offset MPU_Base + 0x800 + (4*n) (MPU_RGDAACn) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	M3PE	M3SM			M3UM r w x				M2PE	M2SM			M2UM r w x				M1PE	M1SM			M1UM r w x				M0PE	M0SM			M0UM r w x			
W																																
Reset (n=0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 212. MPU RGD Alternate Access Control n (MPU_RGDAACn)

Since the MPU_RGDAACn register is simply another memory mapping for MPU_RGDn.Word2, the field definitions shown in *Table 215* are identical to those presented in *Table 213*.

Table 215. MPU_RGDAACn field descriptions

Field	Description
M3PE	Bus master 3 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M3SM	Bus master 3 supervisor mode access control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 0b00 `r, w, x` = read, write and execute allowed 0b01 `r, –, x` = read and execute allowed, but no write 0b10 `r, w, –` = read and write allowed, but no execute 0b11 Same access controls as that defined by M3UM for user mode
M3UM	Bus master 3 user mode access control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	Bus master 2 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M2SM	Bus master 2 supervisor mode access control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 0b00 `r, w, x` = read, write and execute allowed 0b01 `r, –, x` = read and execute allowed, but no write 0b10 `r, w, –` = read and write allowed, but no execute 0b11 Same access controls as that defined by M2UM for user mode

Table 215. MPU_RGDAACn field descriptions (continued)

Field	Description
M2UM	Bus master 2 user mode access control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus master 1 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus master 1 supervisor mode access control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M1UM for user mode
M1UM	Bus master 1 user mode access control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, –, x = read and execute allowed, but no write 0b10 r, w, – = read and write allowed, but no execute 0b11 Same access controls as that defined by M0UM for user mode
M0UM	Bus master 0 user mode access control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

18.7 Functional description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated AHB bus cycles.

18.7.1 Access evaluation macro

As previously discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in *Figure 213*, the access evaluation macro inputs the AHB system bus address phase signals (AHB_ap) and the contents of a region descriptor (RGDn) and performs two major functions: region hit determination (*hit_b*) and detection of an access protection violation (*error*).

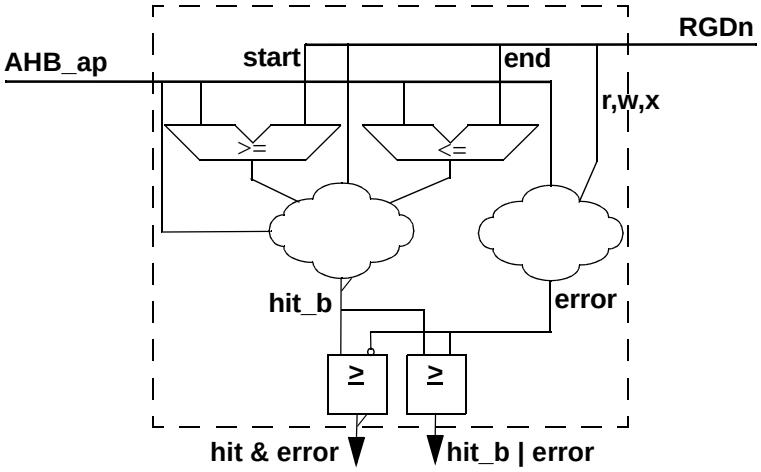


Figure 213. MPU access evaluation macro

Figure 213 is not intended to be a schematic of the actual access evaluation macro, but rather a generalized block diagram showing the major functions included in this logic block.

Access evaluation – hit determination

To determine if the current AHB reference hits in the given region, two magnitude comparators are used in conjunction with the region's start and end addresses. The boolean equation for this portion of the hit determination is defined as:

Equation 13 region_hit

=((haddr[0:26]>= rgdn.srtaddr[0:26]) & (haddr[0:26] <= rgdn.endaddr[0:26]))

& rgdn.vld

where `haddr[*]` is the current AHB reference address, `rgdn.srtaddr[*]` and `rgdn.endaddr[*]` are the start and end addresses, and `rgdn.vld` is the valid bit, all from region descriptor n. Recall there are *no hardware checks* to verify that `rgdn.endaddr` $\geq$ `rgdn.srtaddr`, and it is software's responsibility to properly load appropriate values into these fields of the region descriptor.

In addition to the algebraic comparison of the AHB reference address versus the region descriptor's start and end addresses, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. Using the `hmaster[*]` number to select the appropriate MxPE field from the region descriptor, a process identifier hit term is formed as:

Equation 14 pid_hit

=~rgdn.mxpe

| ((current_pid[0:7] | rgdn.pidmask[0:7])== (rgdn.pid[0:7] | rgdn.pidmask[0:7]))

where the `current_pid[*]` is the selected process identifier from the current bus master, and `rgdn.pid[*]` and `rgdn.pidmask[*]` are the appropriate process identifier fields

from the region descriptor n. For AHB bus masters that do *not* output a process identifier, the MPU forces the `pid_hit` term to be asserted.

As shown in *Figure 213*, the access evaluation macro actually forms the logical complement (*hit_b*) of the combined region_hit and pid_hit boolean equations.

Access evaluation – privilege violation determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the AHB `hmaster[*]` and `hprot[1]` (supervisor/user mode) signals, a set of effective permissions (`eff_rgd[r,w,x]`) is generated from the appropriate fields in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in *Table 216*.

Table 216. Protection violation definition

Description	Inputs					Output
	hwrite	hprot[0]	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection Violation?
inst fetch read	0	0	—	—	0	yes, no x permission
inst fetch read	0	0	—	—	1	no, access is allowed
data read	0	1	0	—	—	yes, no r permission
data read	0	1	1	—	—	no, access is allowed
data write	1	—	—	0	—	yes, no w permission
data write	1	—	—	1	—	no, access is allowed

The resulting boolean equation for the processor protection violations is:

Equation 15 cpu_protection_violation

= ~hwrite & ~hprot[0] & ~eff_rgdn[x]// ifetch & no x

| ~hwrite & hprot[0] & ~eff_rgdn[r]// data_read & no r

| hwrite& ~eff_rgdn[w] // data_write& no w

The resulting boolean equation for the non-processor protection violations is:

Equation 16 protection_violation

= ~hwrite & ~eff_rgdn[r]// data_read & no r

| hwrite& ~eff_rgdn[w] // data_write& no w

As shown in *Figure 213*, the output of the protection violation logic is the *`error`* signal, that is, *`error`* `= protection_violation`.

The access evaluation macro then uses the *`hit_b`* and *`error`* signals to form two outputs. The combined *`(hit_b | error)`* signal is used to signal the current access is not allowed

and *(~hit_b & error)* is used as the input to MPU_EDRn (error detail register) in the event of an error.

The critical timing arc through the access evaluation macro involves the delay from the arrival of `haddr[*]` through the two magnitude comparators and through the *hit_b* generation as shown in *Figure 214*.

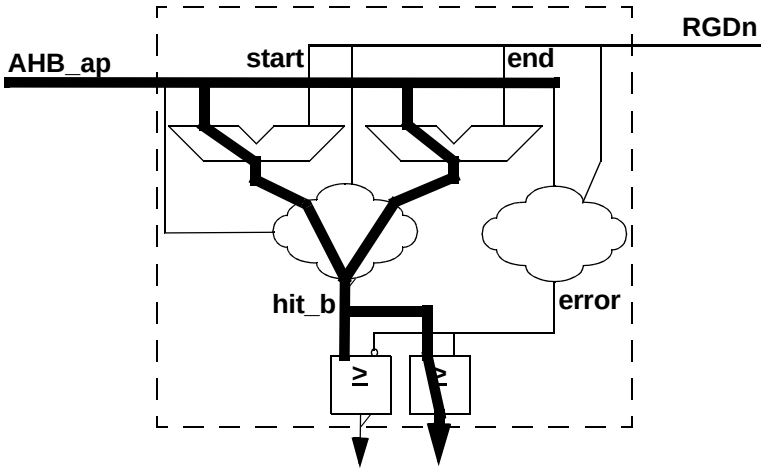


Figure 214. Access evaluation macro critical timing path

18.7.2 Putting it all together and AHB error terminations

For each AHB slave port being monitored, the MPU performs a **reduction-AND** of all the individual *(hit_b | error)* terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in *any* region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, then a protection error is reported.
3. If the access hits in multiple (overlapping) regions and *all* regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to *permission granting over access denying* for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see *Section 18.9, Application information*.

The handling of the AHB bus cycle with a protection error requires two distinct actions. First, recall the protection error logic resides within the AHB address phase pipeline stage. In the event of a protection error, the reference must be inhibited from entering the AHB data phase for the targeted slave device. Stated differently, the reference's address phase must be aborted as viewed by the targeted slave device. This is accomplished by dynamically revising the `htrans[*]` signal sent to the slave device. The `htrans[*]` attribute signals an idle or valid reference (as a non-sequential or sequential access), and this signal is "intercepted" by the MPU. If the access is allowed, then the MPU simply passes the original `htrans[*]` value to the slave device. However if a protection error is detected, then the MPU forces `htrans[*]` sent to the slave to the IDLE encoding. This effectively cancels the transaction before it is seen by the slave device.

While forcing `htrans[*] = IDLE` effectively prevents the bus cycle from being transmitted to the slave device, the AHB transaction has already been “committed” in other portions of the platform, namely in the initiating bus master and the crossbar switch. To properly terminate the bus cycle for these modules, it is necessary to post the standard 2-cycle AHB error response. For this response, the `hresp[*]` signal is forced to the ERROR encoding for 2 cycles, the first with `hready` negated and the second with `hready` asserted. To perform these termination functions, the MPU “intercepts” the `hready` and `hresp[*]` signals from the slave devices, performs the required logic functions to force the response on the bus cycle with a protection error, and then outputs the adjusted values to the crossbar switch.

18.8 Initialization information

The reset state of MPU_CESR[VLD] disables the entire module. Recall while the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU_RGDn) are loaded at system startup, including the setting of the MPU_RGDn.Word3[VLD] bits, *before* MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Recall if a memory reference does not hit in *any* region descriptor, the attempted access is terminated with an error.

18.9 Application information

In an operational system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGDn, it would typically be performed using four 32-bit word writes. As discussed in *Section , MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)*, the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed simply by clearing MPU_RGDn.Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU_RGDAACn) would typically be performed. Recall writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
3. If the region’s start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU_RGDn.Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses respectively and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU’s programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would

be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.

5. When the MPU detects an access error, the current AHB bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EARn and MPU_EDRn registers. The error-terminated AHB bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor can retrieve the captured error address and detail information simply be reading the MPU_E{A,D}Rn registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[SPERR].
6. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can often reduce the number of descriptors required for a given set of access controls. It is important to note that, in the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four bus masters: the two processors (CP0, CP1) and two DMA engines (DMA1, a traditional data movement engine transferring data between RAM and peripherals and DMA2, a second engine transferring data to/from the RAM only).
   Consider the region descriptor assignments in *Table 217*. In this example, there are 8 descriptors used to span 9 regions in the three main spaces of the system memory map (flash, RAM and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

   Of particular interest are the two overlapping spaces: region descriptors 2 & 3 and 3 & 4.

   The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (`rw- | r--`) = (`rw-`) permissions, while CP1 has (`--- | r--`) = (`r--`) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (`r-- | ---`) = (`r--`) permission, while CP1 has (`rw- | r--`) = (`rw-`) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

   The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the IPS bus is partitioned into two regions: one containing the MPU's programming model accessible only to the two processor cores and the remaining peripheral region accessible to both processors and the traditional DMA1 master.

   This simple example is intended to show one possible application of the capabilities of the Memory Protection Unit in a typical system.

Table 217. Overlapping region descriptor example

Region description	RGDn	CP0	CP1	DMA1	DMA2	System space
CP0 code	0	rwx	r--	--	--	Flash memory
CP1 code	1	r--	rwx	--	--	

Table 217. Overlapping region descriptor example (continued)

Region description	RGDn	CP0	CP1	DMA1	DMA2	System space
CP0 data and stack	2	rw-	---	--	--	SRAM
CP0 → CP1 shared data	3	r--	r--	--	--	
CP1 → CP0 shared data						
CP1 data and stack	4	---	rw-	--	--	
Shared DMA data	5	rw-	rw-	rw	rw	
MPU	6	rw-	rw-	--	--	IPS
Peripherals	7	rw-	rw-	rw	--	

19 Enhanced Direct Memory Access (eDMA)

19.1 Introduction

This chapter describes the enhanced Direct Memory Access (eDMA) Controller, a second-generation module capable of performing complex data transfers with minimal intervention from a host processor.

19.2 Overview

The enhanced direct memory access (eDMA) controller hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with SRAM-based local memory containing the transfer control descriptors (TCD) for the channels.

Figure 215 is a block diagram of the eDMA module.

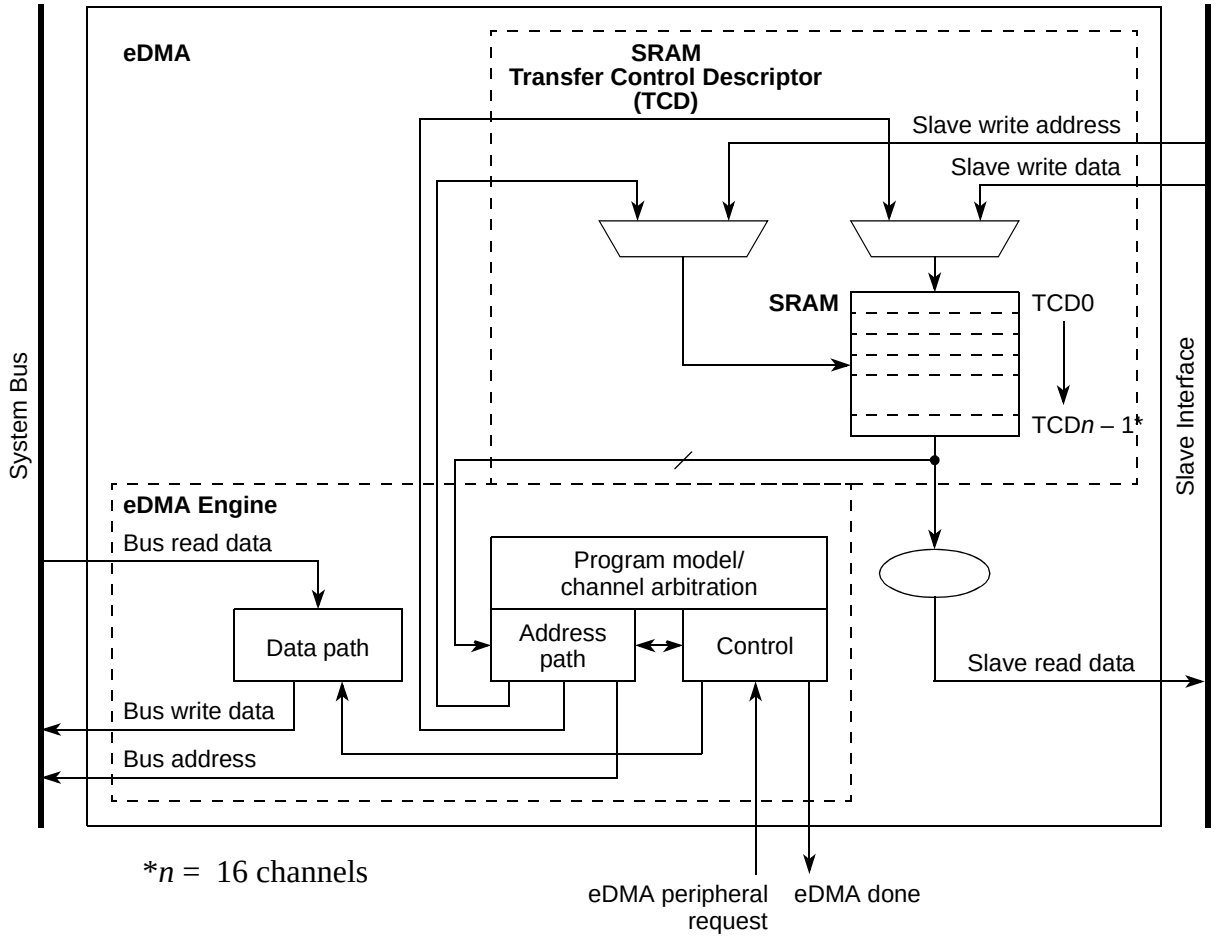


Figure 215. eDMA block diagram

19.3 Features

The eDMA is a highly programmable data transfer engine, which has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is *not* defined within the data packet itself. The eDMA module features:

- All data movement via dual-address transfers: read from source, write to destination
- Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- 16-channel implementation performs complex data transfers with minimal intervention from a host processor
  - 32 bytes of data registers, used as temporary storage to support burst transfers (refer to SSIZE bit)
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD per channel stored in local memory
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continual transfers
  - Peripheral-paced hardware requests (one per channel)

Note: For all three methods, one activation per execution of the minor loop is required.

- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - 1 interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are enabled per channel, and logically summed together to form a single error interrupt.
- Support for scatter/gather DMA processing
- Any channel can be programmed so that it can be suspended by a higher priority channel's activation, before completion of a minor loop.

Throughout this chapter, *n* is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit).

19.4 Modes of operation

19.4.1 Normal mode

In normal mode, the eDMA transfers data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

19.4.2 Debug mode

If enabled by EDMA_CR[EDBG] and the CPU enters debug mode, the eDMA does not grant a service request when the debug input signal is asserted. If the signal asserts during a data block transfer as described by a minor loop in the current active channel's TCD, the eDMA continues the operation until the minor loop completes.

19.5 Memory map and register definition

19.5.1 Memory map

The eDMA programming model is partitioned into two regions:

Region 1 defines control registers; Region 2 defines the local transfer control for the descriptor memory.

Table 218 is a 32-bit view of the eDMA memory map.

Table 218. eDMA memory map

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x0000	EDMA_CR—Control Register	*on page 467*
0x0004	EDMA_ESR—eDMA Error Status Register	*on page 467*
0x0008	Reserved	
0x000C	EDMA_ERQL—eDMA Enable Request Register	*on page 470*
0x0010	Reserved	
0x0014	EDMA_EEIRL—eDMA Enable Error Interrupt Register	*on page 471*
0x0018	EDMA_SERQR—eDMA Set Enable Request Register	*on page 472*
0x0019	EDMA_CERQR—eDMA Clear Enable Request Register	*on page 473*
0x001A	EDMA_SEEI—eDMA Set Enable Error Interrupt Register	*on page 473*
0x001B	EDMA_CEEI—eDMA Clear Enable Error Interrupt Register	*on page 474*
0x001C	EDMA_CIRQR—eDMA Clear Interrupt Request Register	*on page 474*
0x001D	EDMA_CER—eDMA Clear Error Register	*on page 475*
0x001E	EDMA_SSBR—eDMA Set START Bit Register	*on page 476*
0x001F	EDMA_CDSBR—eDMA Clear DONE Status Register	*on page 476*
0x0020	Reserved	
0x0024	EDMA_IRQRL—eDMA Interrupt Request Register	*on page 477*

Table 218. eDMA memory map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x0028	Reserved	
0x002C	EDMA_ERL—eDMA Error Register	on page 478
0x0030	Reserved	
0x0034	EDMA_HRSL—eDMA Hardware Request Status Register	on page 478
0x0038–0x00FF	Reserved	
0x0100	EDMA_CPR0—eDMA Channel 0 Priority Register	on page 479
0x0101	EDMA_CPR1—eDMA Channel 1 Priority Register	on page 479
0x0102	EDMA_CPR2—eDMA Channel 2 Priority Register	on page 479
0x0103	EDMA_CPR3—eDMA Channel 3 Priority Register	on page 479
0x0104	EDMA_CPR4—eDMA Channel 4 Priority Register	on page 479
0x0105	EDMA_CPR5—eDMA Channel 5 Priority Register	on page 479
0x0106	EDMA_CPR6—eDMA Channel 6 Priority Register	on page 479
0x0107	EDMA_CPR7—eDMA Channel 7 Priority Register	on page 479
0x0108	EDMA_CPR8—eDMA Channel 8 Priority Register	on page 479
0x0109	EDMA_CPR9—eDMA Channel 9 Priority Register	on page 479
0x010A	EDMA_CPR10—eDMA Channel 10 Priority Register	on page 479
0x010B	EDMA_CPR11—eDMA Channel 11 Priority Register	on page 479
0x010C	EDMA_CPR12—eDMA Channel 12 Priority Register	on page 479
0x010D	EDMA_CPR13—eDMA Channel 13 Priority Register	on page 479
0x010E	EDMA_CPR14—eDMA Channel 14 Priority Register	on page 479
0x010F	EDMA_CPR15—eDMA Channel 15 Priority Register	on page 479
0x0110–0x0FFF	Reserved	
0x1000	TCD00—Transfer Control Descriptor 0	on page 480
0x1020	TCD01—Transfer Control Descriptor 1	on page 480
0x1040	TCD02—Transfer Control Descriptor 2	on page 480
0x1060	TCD03—Transfer Control Descriptor 3	on page 480
0x1080	TCD04—Transfer Control Descriptor 4	on page 480
0x10A0	TCD05—Transfer Control Descriptor 5	on page 480
0x10C0	TCD06—Transfer Control Descriptor 6	on page 480

Table 218. eDMA memory map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x10E0	TCD07—Transfer Control Descriptor 7	on page 480
0x1100	TCD08—Transfer Control Descriptor 8	on page 480
0x1120	TCD09—Transfer Control Descriptor 9	on page 480
0x1140	TCD10—Transfer Control Descriptor 10	on page 480
0x1160	TCD11—Transfer Control Descriptor 11	on page 480
0x1180	TCD12—Transfer Control Descriptor 12	on page 480
0x11A0	TCD13—Transfer Control Descriptor 13	on page 480
0x11C0	TCD14—Transfer Control Descriptor 14	on page 480
0x11E0	TCD15—Transfer Control Descriptor 15	on page 480
0x1200–0x3FFF	Reserved	

19.5.2 Register descriptions

Read operations on reserved bits in a register return undefined data. Do not write operations to reserved bits. Writing to reserved bits in a register can generate errors. The maximum register bit-width for this device is 16 bits wide.

eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests in one group of 16 channels.

Arbitration can be configured to use either fixed-priority or round-robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 15 down to channel 0, without regard to priority.

Refer to *Section : eDMA Channel n Priority Registers (EDMA_CPRn).*

Figure 216. eDMA Control Register (EDMA_CR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 219. EDMA_CR field descriptions

Field	Description
0-28	Reserved.
29 ERCA	Enable round-robin channel arbitration. 0 Fixed-priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
30 EDBG	Enable debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.
31	Reserved.

eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. For either type of priority configuration error, the ERRCHN field is undefined. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled.

If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit does not equal the TCD.BITER.E_LINK bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the eDMA engine to stop the active channel, and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are *not* affected when an error is detected. After the error status has been updated, the eDMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel executes and terminates with the same error condition.

Figure 217. eDMA Error Status Register (EDMA_ESR)

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 220. EDMA_ESR field descriptions

Field	Description
0 VLD	Logical OR of all EDMA_ERH and EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
1–15	Reserved.
16 GPE	Group priority error. 0 No group priority error. 1 The last recorded error was a configuration error among the group priorities indicating not all group priorities are unique.
17 CPE	Channel priority error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
18–23 ERRCHN[0:5]	Error channel number. Channel number of the last recorded error (excluding GPE and CPE errors). Do not rely on the number in the ERRCHN field for group and channel priority errors. Group and channel priority errors need to be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.
24 SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.SADDR field, indicating TCD.SADDR is inconsistent with TCD.SSIZE.

Table 220. EDMA_ESR field descriptions (continued)

Field	Description
25 SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.SOFF field, indicating TCD.SOFF is inconsistent with TCD.SSIZE.
26 DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.DADDR field, indicating TCD.DADDR is inconsistent with TCD.DSIZE.
27 DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.DOFF field, indicating TCD.DOFF is inconsistent with TCD.DSIZE.
28 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD.NBYTES or TCD.CITER fields, indicating the following conditions exist: – TCD.NBYTES is not a multiple of TCD.SSIZE and TCD.DSIZE, or – TCD.CITER is equal to zero, or – TCD.CITER.E_LINK is not equal to TCD.BITER.E_LINK.
29 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.DLAST_SGA field, indicating TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.E_SG is enabled.
30 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
31 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

eDMA Enable Request Register (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 16 implemented channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 15–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR are provided so that the request enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_ERQRL.

Both the DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request.

Figure 218. eDMA Enable Request Register (EDMA_ERQRL)

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 221. EDMA_ERQRL field descriptions

Field	Description
16–31 ERQ*n*	Enable DMA hardware service request *n.* 0 The DMA request signal for channel n is disabled. 1 The DMA request signal for channel n is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that can affect the ending state of the EDMA_ERQR bit for that channel. If the TCD.D_REQ bit is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the DMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

eDMA Enable Error Interrupt Register (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 16 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 15-0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR are provided so that the error interrupt enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_EEIRL.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Figure 219. eDMA Enable Error Interrupt Register (EDMA_EEIRL)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 222. EDMA_EEIRL field descriptions

Field	Description
16-31 EEI*n*	Enable error interrupt *n.* 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRL to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be set. Setting bit 1 (SERQ*n*) provides a global set function, forcing the entire contents of EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

Figure 220. eDMA Set Enable Request Register (EDMA_SERQR)

Address: Base + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 223. EDMA_SERQR field descriptions

Field	Descriptions
0	Reserved.
1–7 SERQ[0:6]	Set enable request. 0–15 Set corresponding bit in EDMA_ERQRL 16–63Reserved 64–127Set all bits in EDMA_ERQRL Bit 2 (SERQ1) is not used.

eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ*n*) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes.

Figure 221. eDMA Clear Enable Request Register (EDMA_CERQR)

Address: Base + 0x0019 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 224. EDMA_CERQR field descriptions

Field	Description
0	Reserved.
1–7 CERQ[0:6]	Clear enable request. 0–15 Clear corresponding bit in EDMA_ERQRL 16–63Reserved 64–127Clear all bits in EDMA_ERQRL Bit 2 (CERQ1) is not used.

eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a simple memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI*n*) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

Figure 222. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Address: Base + 0x001A Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 225. EDMA_SEEIR field descriptions

Field	Description
0	Reserved.
1–7 SEEI[0:6]	Set enable error interrupt. 0–15 Set corresponding bit in EDMA_EIRRL 16–63 Reserved 64–127 Set all bits in EDMA_EEIRL Bit 2 (SEEI1) is not used.

eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI*n*) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

Figure 223. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Address: Base + 0x001B Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 226. EDMA_CEEIR field descriptions

Field	Description
0	Reserved.
1–7 CEEI[0:6]	Clear enable error interrupt. 0–15 Clear corresponding bit in EDMA_EEIRL 16–63 Reserved 64–127 Clear all bits in EDMA_EEIRL Bit 2 (CEEI1) is not used.

eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT*n*) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Figure 224. eDMA Clear Interrupt Request (EDMA_CIRQR)

Address: Base + 0x001C Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 227. EDMA_CIRQR field descriptions

Field	Description
0	Reserved.
1–7 CINT[0:6]	Clear interrupt request. 0–15 Clear corresponding bit in EDMA_IRQRL 16–63 Reserved 64–127 Clear all bits in EDMA_IRQRL Bit 2 (CINT1) is not used.

eDMA Clear Error Register (EDMA_CER)

The EDMA_CER provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CER*n*) provides a global clear function, forcing the entire contents of the EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

Figure 225. eDMA Clear Error Register (EDMA_CER)

Address: Base + 0x001D Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CER[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 228. EDMA_CER field descriptions

Field	Description
0	Reserved.
1–7 CER[0:6]	Clear error indicator. 0–15 Clear corresponding bit in EDMA_ERL 16–63 Reserved 64–127 Clear all bits in EDMA_ERL

eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB*n*) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Figure 226. eDMA Set START Bit Register (EDMA_SSBR)

Address: Base + 0x001E Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 229. EDMA_SSBR field descriptions

Field	Description
0	Reserved.
1–7 SSB[0:6]	Set START bit (channel service request). 0–15 Set the corresponding channel's TCD START bit 16–63 Reserved 64–127 Set all TCD START bits Bit 2 (SSB1) is not used.

eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB*n*) provides a global clear function, forcing all DONE bits to be cleared.

Figure 227. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

Address: Base + 0x001F Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 230. EDMA_CDSBR field descriptions

Field	Description
0	Reserved.
1–7 CDSB[0:6]	Clear DONE status bit. 0–15 Clear the corresponding channel's DONE bit 16–63 Reserved 64–127 Clear all TCD DONE bits Bit 2 (CDSB1) is not used.

eDMA Interrupt Request Register (EDMA_IRQRL)

The EDMA_IRQRL provide a bit map for the 16 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 15–0.

The eDMA engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, it is software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a *single* channel can easily be cleared without the need to perform a read-modify-write sequence to the EDMA_IRQRL.

Figure 228. eDMA Interrupt Request Low Register (EDMA_IRQRL)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT 15	INT 14	INT 13	INT 12	INT 11	INT 10	INT 09	INT 08	INT 07	INT 06	INT 05	INT 04	INT 03	INT 02	INT 01	INT 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 231. EDMA_IRQRL field descriptions

Field	Description
16–31 INT*n*	eDMA interrupt request *n.* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

eDMA Error Register (EDMA_ERL)

The EDMA_ERL provides a bit map for the 16 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 15-0.

The eDMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIR, then logically summed across groups of 16 and 32 channels to form several group error interrupt requests that are then routed to the interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIR. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CER. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no affect on the corresponding channel's current error status. The EDMA_CER is provided so the error indicator for a *single* channel can easily be cleared.

Figure 229. eDMA Error Low Register (EDMA_ERL)

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 232. EDMA_ERL field descriptions

Field	Description
16–31 ERR*n*	eDMA Error *n.* 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

DMA Hardware Request Status (DMAHRSL)

The DMAHRSL registers provide a bit map for the implemented channels 16 to show the current hardware request status for each channel. DMAHRSL covers channels 31:00. Hardware request status reflects the current state of the registered and qualified (via the DMAERQ field) ipd_req lines as seen by the eDMA's arbitration logic. This view into the hardware request signals may be used for debug purposes.

Figure 230. EDMA Hardware Request Status Register Low (EDMA_HRSL)

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS	HRS
W	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 233. EDMA_HRSL field descriptions

Field	Description
16–31 HRSn	DMA Hardware Request Status 0 A hardware service request for channel n is not present. 1 A hardware service request for channel n is present. The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

eDMA Channel n Priority Registers (EDMA_CPRn)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software chooses to modify channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the EDMA_CPR*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the EDMA_CPR*n* registers. The group priority is assigned in the EDMA_CR.

Refer to *Figure 216* and *Table 219* for the EDMA_CR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR*n* register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes.

Figure 231. eDMA Channel *n* Priority Register (EDMA_CPR*n*)

Address: Base + 0x100 + *n* Access: User read/write

	0	1	2	3	4	5	6	7
R	ECP	0	0	0	CHPRI			
W								
Reset	0	0	0	0	— [1]			

1. The reset value for the channel priority fields, GRPPRI[0–1] and CHPRI[0–3] is the channel number for the priority register; EDMA_CPR15[CHPRI] = 0b1111.

The following table describes the fields in the eDMA channel *n* priority register:

Table 234. EDMA_CPR*n* field descriptions

Field	Description
0 ECP	Enable channel preemption. 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
1-3	Reserved.
4–7 CHPRI[0:3]	Channel *n* arbitration priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

Transfer Control Descriptor (TCD)

Each channel requires a 256-bit transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as 23 variable-length fields.

Table 235 defines the fields of the basic TCD structure.

Table 235. TCD*n* 32-bit memory structure

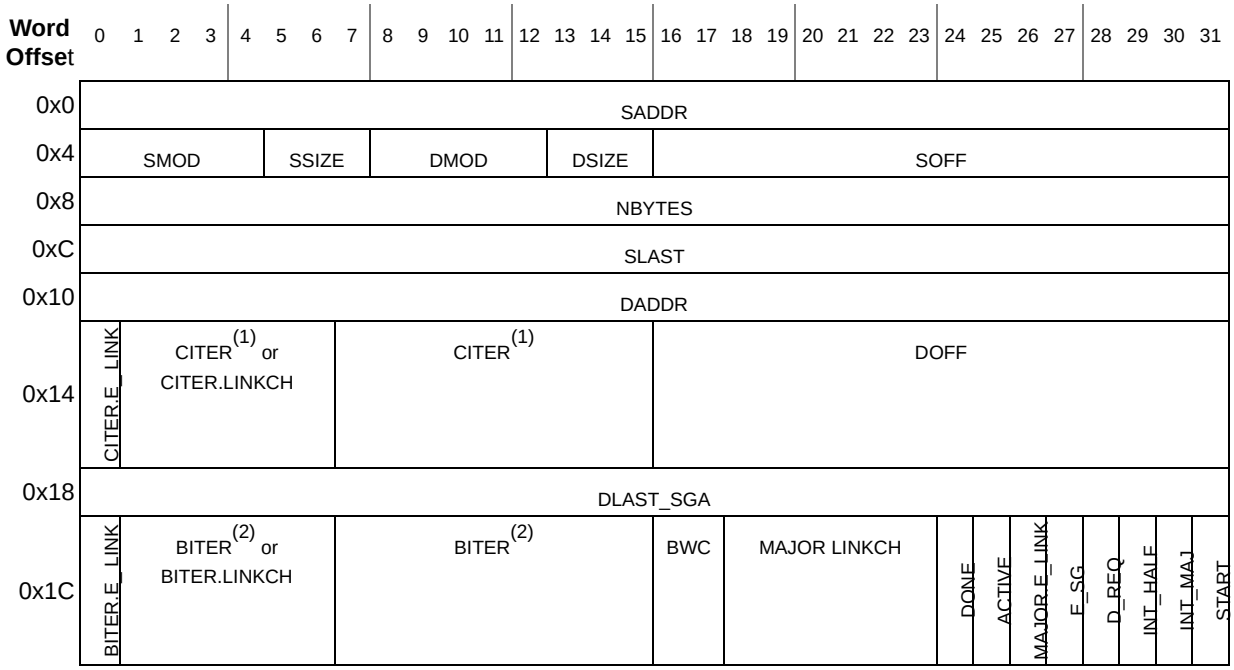
eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 × *n*) + 0	32	Source address	SADDR	Word 0
0x1000 + (32 × *n*) + 32	5	Source address modulo	SMOD	Word 1
0x1000 + (32 × *n*) + 37	3	Source data transfer size	SSIZE	
0x1000 + (32 × *n*) + 40	5	Destination address modulo	DMOD	
0x1000 + (32 × *n*) + 45	3	Destination data transfer size	DSIZE	
0x1000 + (32 × *n*) + 48	16	Signed Source Address Offset	SOFF	
0x1000 + (32 × *n*) + 64	32	Inner minor byte count	NBYTES	Word 2
0x1000 + (32 × *n*) + 96	32	Last Source Address Adjustment	SLAST	Word 3
0x1000 + (32 × *n*) + 128	32	Destination Address	DADDR	Word 4

Table 235. TCD*n* 32-bit memory structure (continued)

eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 × *n*) + 160	1	Channel-to-channel Linking on Minor Loop Complete	CITER.E_LINK	Word 5
0x1000 + (32 × *n*) + 161	6	Current Major Iteration Count or Link Channel Number	CITER or CITER.LINKCH	
0x1000 + (32 × *n*) + 167	9	Current Major Iteration Count	CITER	
0x1000 + (32 × *n*) + 176	16	Destination Address Offset (Signed)	DOFF	
0x1000 + (32 × *n*) + 192	32	Last Destination Address Adjustment / Scatter Gather Address	DLAST_SGA	Word 6
0x1000 + (32 × *n*) + 224	1	Channel-to-channel Linking on Minor Loop Complete	BITER.E_LINK	Word 7
0x1000 + (32 × *n*) + 225	6	Starting Major Iteration Count or Link Channel Number	BITER or BITER.LINKCH	
0x1000 + (32 × *n*) + 231	9	Starting Major Iteration Count	BITER	
0x1000 + (32 × *n*) +240	2	Bandwidth Control	BWC	
0x1000 + (32 × *n*) + 242	6	Link Channel Number	MAJOR.LINKCH	
0x1000 + (32 × *n*) + 248	1	Channel Done	DONE	
0x1000 + (32 × *n*) + 249	1	Channel Active	ACTIVE	
0x1000 + (32 × *n*) + 250	1	Channel-to-channel Linking on Major Loop Complete	MAJOR.E_LINK	
0x1000 + (32 × *n*) + 251	1	Enable Scatter/Gather Processing	E_SG	
0x1000 + (32 × *n*) + 252	1	Disable Request	D_REQ	
0x1000 + (32 × *n*) + 253	1	Channel Interrupt Enable When Current Major Iteration Count is Half Complete	INT_HALF	
0x1000 + (32 × *n*) + 254	1	Channel Interrupt Enable When Current Major Iteration Count Complete	INT_MAJ	
0x1000 + (32 × *n*) + 255	1	Channel Start	START	

Figure 232 defines the fields of the TCD*n* structure.

Figure 232. TCD structure



1. If channel linking on minor link completion is disabled, TCD bits [161:175] form a 15-bit CITER field; if channel-to-channel linking is enabled, CITER becomes a 9-bit field.
2. If channel linking on minor link completion is disabled, TCD bits [225:239] form a 15-bit BITER field; if channel-to-channel linking is enabled, BITER becomes a 9-bit field.

Note: The TCD structures for the eDMA channels shown in Figure 232 are implemented in internal SRAM. These structures are not initialized at reset. Therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 236 gives a detailed description of the TCN*n* fields.

Table 236. TCD*n* field descriptions

Bits Word Offset [n:n]	Field Name	Description
0–31 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. not 0 This value defines a specific address range that is specified to be either the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, start the queue at a 0-modulo-size address and set the SMOD field to the value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.

Table 236. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
37–39 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 32-bit 101 32-byte burst (64-bit x 4) 110 Reserved 111 Reserved The attempted specification of a ‘reserved’ encoding causes a configuration error.
40–44 0x4 [8:12]	DMOD [0:4]	Destination address modulo. Refer to the SMOD[0:5] definition.
45–47 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. Refer to the SSIZE[0:2] definition.
48–63 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64–95 0x8 [0:31]	NBYTES [0:31]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a four GB transfer.
96–127 0xC [0:31]	SLAST [0:31]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer 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.
128–159 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. This bit must be equal to the BITER.E_LINK bit otherwise a configuration error is reported.

Table 236. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
161–166 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] form a 15-bit CITER field. otherwise – After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel’s TCD.START bit.
167–175 0x14 [7:15]	CITER [6:14]	Current “major” iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
176–191 0x14 [16:31]	DOFF [0:15]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
192–223 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If scatter/gather processing for the channel is disabled (TCD.E_SG = 0) then – Adjustment value added to the destination address at the completion of the outer 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. Otherwise – This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This 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.

Table 236. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
224 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
225–230 0x1C [1:6]	BITER [0:5] or BITER.LINKCH [0:5]	Beginning or starting “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] form a 15-bit BITER field. Otherwise – After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel’s TCD.START bit. When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 0x1C [7:15]	BITER [6:14]	Beginning or starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
240–241 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). To minimize start-up latency, bandwidth control stalls are suppressed for the first two system bus cycles and after the last write of each minor loop. 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for four cycles after each r/w 11 eDMA engine stalls for eight cycles after each r/w

Table 236. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
242–247 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then: – No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise – After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 0x1C [24]	DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the eDMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the channel has begun to be processed by the eDMA engine, not when the first data transfer occurs). This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 0x1C [25]	ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the inner minor loop completes or if any error condition is detected.
250 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. NOTE: To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 0x1C [27]	E_SG	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure that is loaded as the transfer control descriptor into the local memory. NOTE: To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
252 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQL bit is cleared when the outer major loop is complete.

Table 236. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
253 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the bit in the EDMA_ERQL when the current major iteration count reaches the halfway point. The eDMA engine performs the compare (CITER == (BITER >> 1)). This halfway point interrupt request supports double-buffered (aka ping-pong) schemes, or where the processor needs an early indication of the data transfer's progress during data movement. CITER = BITER = 1 with INT_HALF enabled generates an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 0x1C [31]	START	Channel start. If this flag is set, the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

19.6 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

19.6.1 eDMA microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules, as shown in the following list:

- eDMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel 'x' and channel 'y,' and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_CPR*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

    When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCDn.{SADDR, DADDR, CITER}

back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter/gather operation.

– Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.

  The address and data path modules directly support the 2-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the 2nd stage of the pipeline (the data phase).

– Program model/channel arbitration: This module implements the first section of eDMA's programming model as well as the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the Control logic).

– Control: This module provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner 'minor loop' byte count has been moved.

  A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as the following:

```
if (SSIZE < DSIZE)
transfer size = destination transfer size (# of bytes)
else
transfer size = source transfer size (# of bytes)
```

  Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

  For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  – Memory controller: This logic implements the required dual-ported controller, handling accesses from both the eDMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
  – Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

19.6.2 eDMA basic data flow

The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 233*, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same

basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed through the eDMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the eDMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.

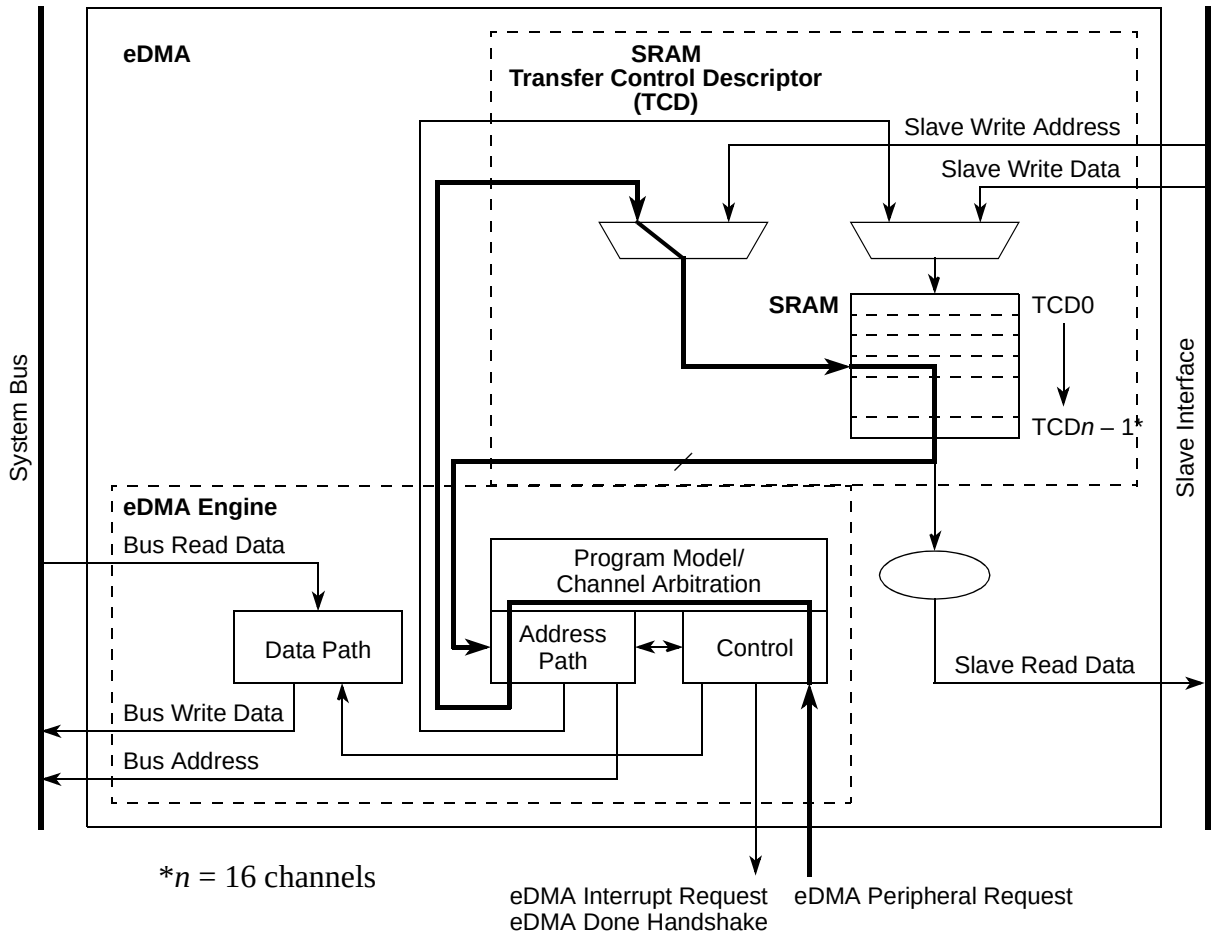


Figure 233. eDMA operation, part 1

In the second part of the basic data flow as shown in *Figure 234*, the modules associated with the data transfer (address path, data path and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA Done Handshake signal is asserted at the end of the minor byte count transfer.

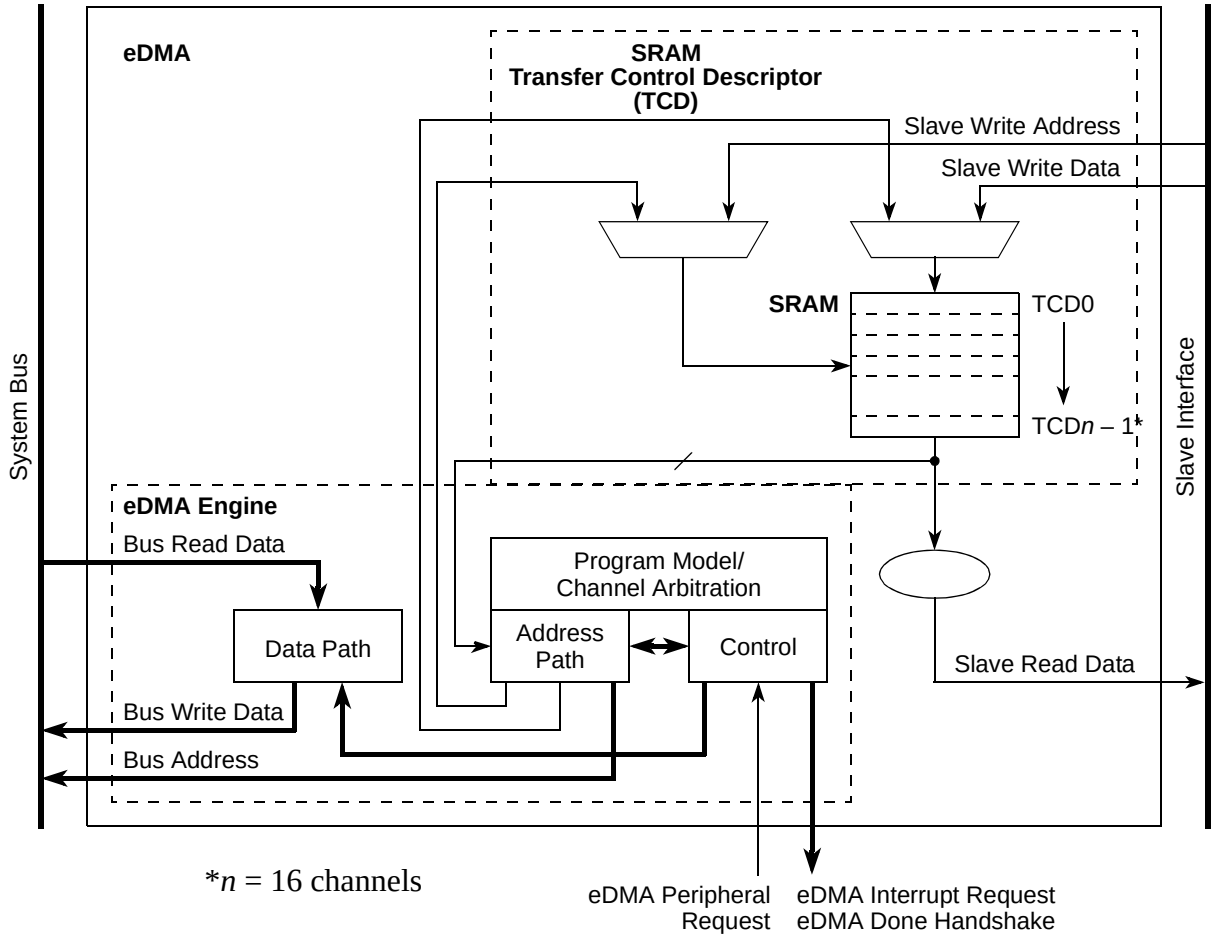


Figure 234. eDMA operation, part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD: for example., SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in *Figure 235*.

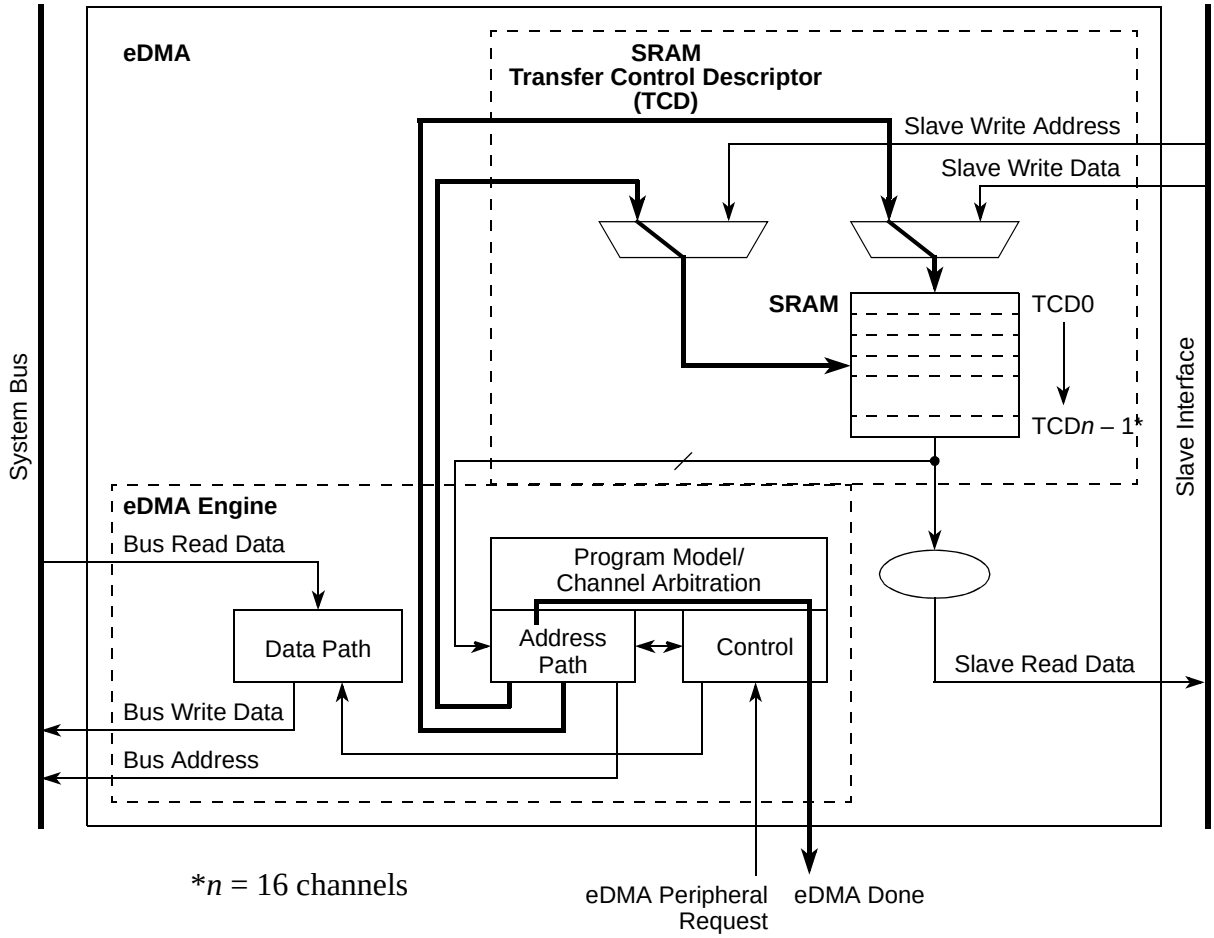


Figure 235. eDMA operation, part 3

19.6.3 eDMA performance

This section addresses the performance of the eDMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the eDMA. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests that can be serviced in a fixed time is a more useful metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the eDMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in *Table 237*. The following assumptions apply to *Table 237* and *Table 238*:

- Internal SRAM can be accessed with zero wait-states when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- All slave accesses are 32-bits in size.

Table 237. eDMA peak transfer rates (MB/Sec)

System Speed, Transfer Size	Internal SRAM-to-Internal SRAM	32-bit Slave-to-Internal SRAM	Internal SRAM-to-32-bit Slave (buffering disabled)	Internal SRAM-to-32-bit Slave (buffering enabled)
66.7 MHz, 32-bit	66.7	66.7	53.3	88.7
66.7 MHz, 64-bit	133.3	66.7	53.3	88.7
66.7 MHz, 256-bit(1)	213.4	N/A(2)	N/A(2)	N/A(2)
83.3 MHz, 32-bit	83.3	83.3	66.7	110.8
83.3 MHz, 64-bit	166.7	83.3	66.7	110.8
83.3 MHz, 256-bit(1)	266.6	N/A(2)	N/A(2)	N/A(2)
100.0 MHz, 32-bit	100.0	100.0	80.0	133.0
100.0 MHz, 64-bit	200.0	100.0	80.0	133.0
100.0 MHz, 256-bit(1)	320.0	N/A(2)	N/A(2)	N/A(2)
132.0 MHz, 32-bit	132.0	132.0	105.6	175.6
132.0 MHz, 64-bit	264.0	132.0	105.6	175.6
132.0 MHz, 256-bit(1)	422.4	N/A(2)	N/A(2)	N/A(2)

1. A 256-bit transfer occurs as a burst of four 64-bit beats.
2. Not applicable: burst access to a slave port is not supported.

Table 237 presents a peak transfer rate comparison, measured in MBs per second where the internal-SRAM-to-internal-SRAM transfers occur at the core's datapath width; that is, either 32- or 64-bits per access. For all transfers involving the slave bus, 32-bit transfer sizes are used. In all cases, the transfer rate includes the time to read the source plus the time to write the destination.

The second performance metric is a measure of the number of DMA requests that can be serviced in a given amount of time. For this metric, it is assumed the peripheral request causes the channel to move a single slave-mapped operand to/from internal SRAM. The same timing assumptions used in the previous example apply to this calculation. In

particular, this metric also reflects the time required to activate the channel. The eDMA design supports the following hardware service request sequence:

- Cycle 1: eDMA peripheral request is asserted.
- Cycle 2: The eDMA peripheral request is registered locally in the eDMA module and qualified. (TCD.START bit initiated requests start at this point with the registering of the slave write to TCD bit 255).
- Cycle 3: Channel arbitration begins.
- Cycle 4: Channel arbitration completes. The transfer control descriptor local memory read is initiated.
- Cycle 5–6: The first two parts of the activated channel's TCD is read from the local memory. The memory width to the eDMA engine is 64 bits, so the entire descriptor can be accessed in four cycles.
- Cycle 7: The first system bus read cycle is initiated, as the third part of the channel's TCD is read from the local memory. Depending on the state of the crossbar switch, arbitration at the system bus can insert an additional cycle of delay here.
- Cycle 8 – *n*: The last part of the TCD is read in. This cycle represents the 1st data phase for the read, and the address phase for the destination write.

The exact timing from this point is a function of the response times for the channel's read and write accesses. In this case of an slave read and internal SRAM write, the combined data phase time is 4 cycles. For an SRAM read and slave write, it is 5 cycles.

- Cycle *n* + 1: This cycle represents the data phase of the last destination write.
- Cycle *n* + 2: The eDMA engine completes the execution of the inner minor loop and prepares to write back the required TCD*n* fields into the local memory. The control/status fields at word offset 0x1C in TCD*n* are read. If the major loop is complete, the MAJOR.E_LINK and E_SG bits are checked and processed if enabled.
- Cycle *n* + 3: The appropriate fields in the first part of the TCD*n* are written back into the local memory.
- Cycle *n* + 4: The fields in the second part of the TCD*n* are written back into the local memory. This cycle coincides with the next channel arbitration cycle start.
- Cycle *n* + 5: The next channel to be activated performs the read of the first part of its TCD from the local memory. This is equivalent to Cycle 4 for the first channel's service request.

Assuming zero wait states on the system bus, DMA requests can be processed every 9 cycles. Assuming an average of the access times associated with slave-to-SRAM (4 cycles) and SRAM-to-slave (5 cycles), DMA requests can be processed every 11.5 cycles (4 + (4 + 5)/2 + 3). This is the time from Cycle 4 to Cycle "*n* 5." The resulting peak request rate, as a function of the system frequency, is shown in *Table 238*. This metric represents millions of requests per second.

Table 238. eDMA peak request Rate (MReq/sec)

System Frequency (MHz)	Request Rate (Zero Wait States)	Request Rate (with Wait States)
66.6	7.4	5.8
83.3	9.2	7.2
100.0	11.1	8.7

Table 238. eDMA peak request Rate (MReq/sec) (continued)

System Frequency (MHz)	Request Rate (Zero Wait States)	Request Rate (with Wait States)
133.3	14.8	11.6
150.0	16.6	13.0

A general formula to compute the peak request rate (with overlapping requests) is:

Equation 17

PEAKreq = freq / [entry + (1 + read_ws) + (1 + write_ws) + exit]

where:

PEAKreq — peak request rate

freq — system frequency

entry — channel startup (four cycles)

read_ws — wait states seen during the system bus read data phase

write_ws — wait states seen during the system bus write data phase

exit — channel shutdown (three cycles)

For example: consider a system with the following characteristics:

- Internal SRAM can be accessed with one wait-state when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- System operates at 150 MHz.

For an SRAM to slave transfer,

Equation 18

PEAKreq = 150 MHz / [4 + (1 + 1) + (1 + 3) + 3] cycles = 11.5 Mreq/sec

For an slave to SRAM transfer,

Equation 19

PEAKreq = 150 MHz / [4 + (1 + 2) + (1 + 1) + 3] cycles = 12.5 Mreq/sec

Assuming an even distribution of the two transfer types, the average peak request rate is:

Equation 20

PEAKreq = (11.5 Mreq/sec + 12.5 Mreq/sec) / 2 = 12.0 Mreq/sec

The minimum number of cycles to perform a single read/write, zero wait states on the system bus, from a cold start (no channel is executing, eDMA is idle) are the following:

- 11 cycles for a software (TCD.START bit) request
- 12 cycles for a hardware (eDMA peripheral request signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the eDMA peripheral request signals. For the peak

request rate calculations above, the arbitration and request registering is absorbed in or overlap the previous executing channel.

Note: When channel linking or scatter/gather is enabled, a two-cycle delay is imposed on the next channel selection and startup. This allows the link channel or the scatter/gather channel to be eligible and considered in the arbitration pool for next channel selection.

19.7 Initialization / application information

19.7.1 eDMA initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers (optional).
4. Write the 32-byte TCD for each channel that can request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by either software (setting the TCD.START bit) or by hardware (slave device asserting its eDMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine reads the entire TCD, including the primary transfer control parameter shown in *Table 239*, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the eDMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed: for example, interrupts, major loop channel linking, and scatter/gather operations, if enabled.

Table 239. TCD primary control and status fields

TCD Field Name	Description
START	Control bit to explicitly start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (Cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for “throttling” bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature

Table 239. TCD primary control and status fields (continued)

TCD Field Name	Description
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 236 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

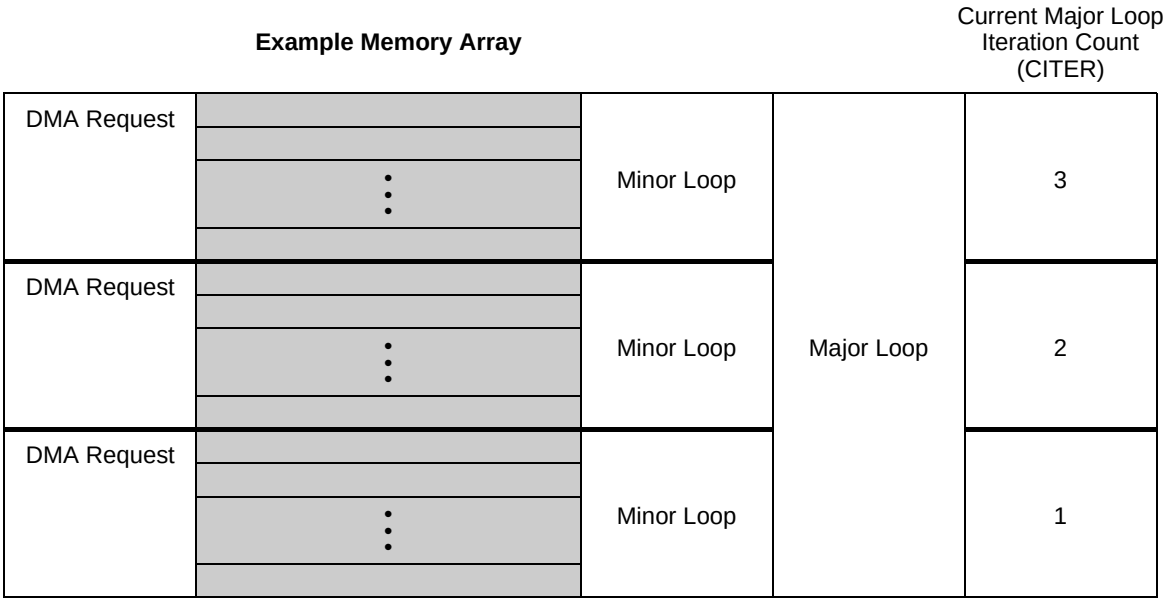


Figure 236. Example of multiple loop iterations

Figure 237 lists the memory array terms and how the TCD settings interrelate.

<table>
<tr><td>xADDR:
(Starting Address)</td><td>xSIZE:
(Size of one data transfer)
•
•
•</td><td>Minor Loop (NBYTES in Minor Loop, often the same value as xSIZE)</td><td rowspan="3">Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE)

Each DMA Source (S) and Destination (D) has its own:
• Address (xADDR)
• Size (xSIZE)
• Offset (xOFF)
• Modulo (xMOD)
• Last Address Adjustment (xLAST) where x = S or D

Peripheral queues typically have size and offset equal to NBYTES</td></tr>
<tr><td>•
•
•</td><td>•
•
•</td><td>Minor Loop</td></tr>
<tr><td>xLAST: Number of bytes added to current address after Major Loop (Typically used to loop back)</td><td>•
•
•</td><td>Last Minor Loop</td></tr>
</table>

Figure 237. Memory array terms

19.7.2 DMA programming errors

The eDMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors: group priority error and channel priority error, or EDMA_ESR[GPE] and EDMA_ESR[CPE], respectively.

For all error types other than group or channel priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

If priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts and error reporting are associated with the selected channel.

19.7.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in *Table 240*. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 240. DMA request summary for eDMA

DMA Request	Ch.	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source

Table 240. DMA request summary for eDMA (continued)

DMA Request	Ch.	Source	Description
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source

19.7.4 DMA arbitration mode considerations

Fixed-channel arbitration

In this mode, the channel service request from the highest priority channel is selected to execute. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

Fixed-group arbitration, round-robin channel arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

19.7.5 DMA transfer

Single request

To perform a simple transfer of ‘*n*’ bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer completes, the TCD.DONE bit is set and an interrupt is generated if correctly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte-wide memory port located at 0x1000. The destination memory has a word-wide port located at 0x2000. The address offsets are programmed in increments

to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

TCD.CITER = TCD.BITER = 1

TCD.NBYTES = 16

TCD.SADDR = 0x1000

TCD.SOFF = 1

TCD.SSIZE = 0

TCD.SLAST = –16

TCD.DADDR = 0x2000

TCD.DOFF = 4

TCD.DSIZE = 2

TCD.DLAST_SGA= –16

TCD.INT_MAJ = 1

TCD.START = 1 (Initialize all other fields before writing to this bit)

All other TCD fields = 0

This generates the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte (0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word (0x2000) → first iteration of the minor loop
   c) read_byte (0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word (0x2004) → second iteration of the minor loop
   e) read_byte (0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
   f) write_word (0x2008) → third iteration of the minor loop
   g) read_byte (0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
   h) write_word (0x200c) → last iteration of the minor loop → major loop complete
6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

Multiple requests

The next example is the same as previous with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the

EDMA_ERQR, channel service requests are initiated by the slave device (set ERQR after TCD; TCD.START = 0).

TCD.CITER = TCD.BITER = 2

TCD.NBYTES = 16

TCD.SADDR = 0x1000

TCD.SOFF = 1

TCD.SSIZE = 0

TCD.SLAST = –32

TCD.DADDR = 0x2000

TCD.DOFF = 4

TCD.DSIZE = 2

TCD.DLAST_SGA = –32

TCD.INT_MAJ = 1

TCD.START = 0 (Initialize all other fields before writing this bit.)

All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers execute as follows:
   a) read_byte (0x1000), read_byte (0x1001), read_byte (0x1002), read_byte (0x1003)
   b) write_word (0x2000) → first iteration of the minor loop
   c) read_byte (0x1004), read_byte (0x1005), read_byte (0x1006), read_byte (0x1007)
   d) write_word (0x2004) → second iteration of the minor loop
   e) read_byte (0x1008), read_byte (0x1009), read_byte (0x100a), read_byte (0x100b)
   f) write_word (0x2008) → third iteration of the minor loop
   g) read_byte (0x100c), read_byte (0x100d), read_byte (0x100e), read_byte (0x100f)
   h) write_word (0x200c) → last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
7. eDMA engine writes: TCD.ACTIVE = 0.
8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers execute as follows:
    a) read_byte (0x1010), read_byte (0x1011), read_byte (0x1012), read_byte (0x1013)
    b) write_word (0x2010) → first iteration of the minor loop
    c) read_byte (0x1014), read_byte (0x1015), read_byte (0x1016), read_byte (0x1017)
    d) write_word (0x2014) → second iteration of the minor loop
    e) read_byte (0x1018), read_byte (0x1019), read_byte (0x101a), read_byte (0x101b)
    f) write_word (0x2018) → third iteration of the minor loop
    g) read_byte (0x101c), read_byte (0x101d), read_byte (0x101e), read_byte (0x101f)
    h) write_word (0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

Modulo feature

The modulo feature of the eDMA provides the ability to easily implement a circular data queue in which the size of the queue is a power of 2. MOD is a 5-bit field for the source and destination in the TCD, and specifies which lower address bits are incremented from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. Clearing this field to 0 disables the modulo feature.

Table 241 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 241. Modulo feature example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

19.7.6 TCD status

Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method can be extracted from the following sequence. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a one. Polling the TCD.ACTIVE bit can be inconclusive because the active status can be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (issued service request via software)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed major loop and is idle)

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (issued service request via hardware)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed major loop and is idle)

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

Active channel TCD reads

the eDMA reads the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

Preemption status

Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel

and/or group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

19.7.7 Channel linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.START bit of another channel (or itself) thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field determines whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last.

When the major loop is exhausted, only the major loop channel link fields are used to determine whether to make a channel link. For example, with the initial fields of:

TCD.CITER.E_LINK = 1

TCD.CITER.LINKCH = 0xC

TCD.CITER value = 0x4

TCD.MAJOR.E_LINK = 1

TCD.MAJOR.LINKCH = 0x7

channel linking executes as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit
3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

Note: After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, half-way done interrupt point.

Table 242 summarizes how a DMA channel can “link” to another DMA channel, i.e, use another channel’s TCD, at the end of a loop.

Table 242. Channel linking parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of Minor Loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration)
	citer.linkch	Link channel number when linking at end of minor loop (current iteration)
Link at end of Major Loop	major.e_link	Enable channel-to-channel linking on major loop completion
	major.linkch	Link channel number when linking at end of major loop

19.7.8 Dynamic programming

This section provides recommended methods to change the programming model during channel execution.

Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits during channel execution. These bits are read from the TCD local memory at the *end* of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.MAJOR.E_LINK bit at the same time the eDMA engine is retiring the channel. The TCD.MAJOR.E_LINK is set in the programmer's model, but it is unclear whether the link completed before the channel retired.

Use the following coherency model when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD.MAJOR.E_LINK bit
2. Read the TCD.MAJOR.E_LINK bit
3. Test the TCD.MAJOR.E_LINK request status:
   a) If the bit is set, the dynamic link attempt was successful.D
   b) If the bit is cleared, the channel had already retired before the dynamic link completed.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.MAJOR.E_LINK and TCD.E_SG bits to zero on any writes to a channel's TCD after that channel's TCD.DONE bit is set indicating the major loop is complete.

Note: The user must clear the TCD.DONE bit before writing the TCD.MAJOR.E_LINK or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

20 DMA Channel Mux (DMA_MUX)

20.1 Introduction

20.1.1 Overview

The DMA Mux allows to route a configurable amount of DMA sources (slots) to a configurable amount of DMA channels. This is illustrated in *Figure 238*.

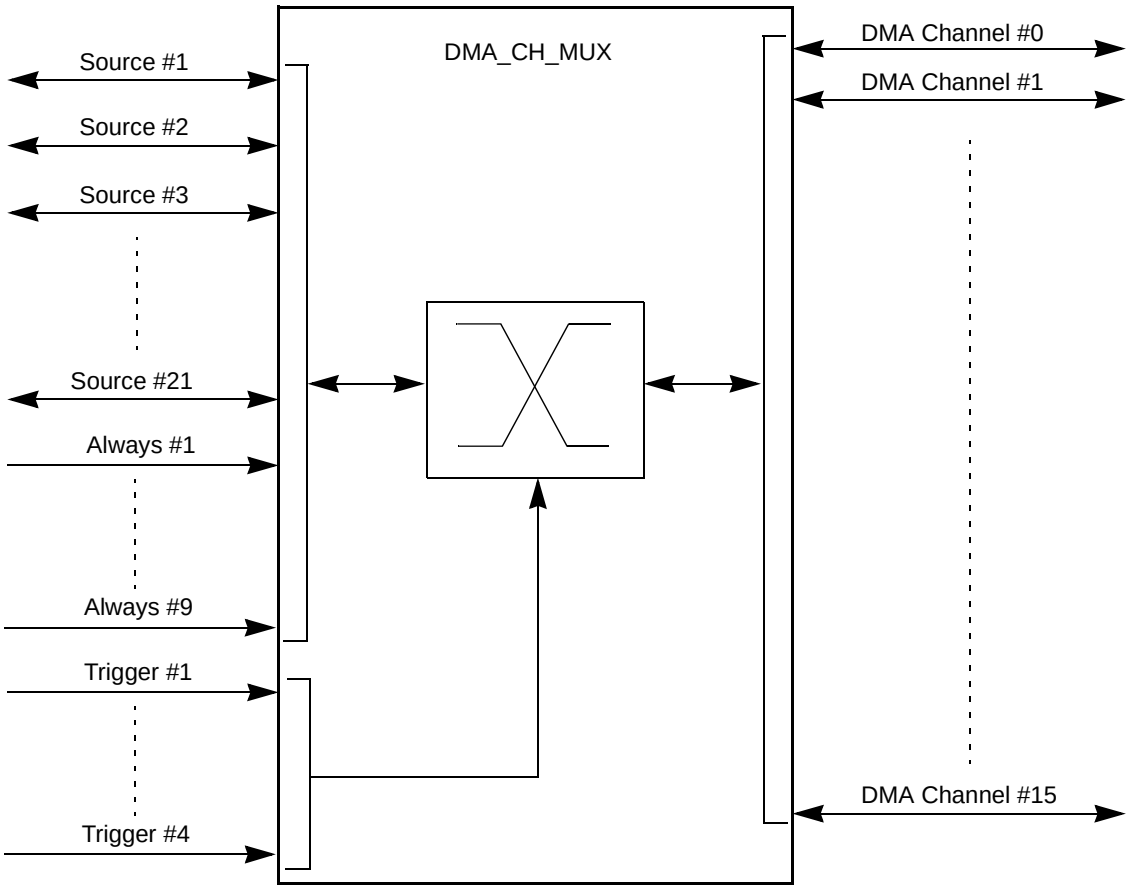


Figure 238. DMA Mux block diagram

20.1.2 Features

The DMA Mux has these major features:

- 16 independently selectable DMA channel routers
  - 4 channels with normal or periodic triggering capability
  - 12 channels with normal capability
- Each channel router can be assigned to 1 of 21 possible peripheral DMA sources

20.1.3 Modes of operation

The following operation modes are available:

- Disabled Mode

  In this mode, the DMA channel is disabled. Since disabling and enabling of DMA channels is done primarily via the DMA configuration registers, this mode is used mainly as the reset state for a DMA channel in the DMA Channel Mux. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place (for example, changing the period of a DMA trigger).

- Normal Mode

  In this mode, a DMA source (such as DSPI_0_TX or DSPI_0_RX example) is routed directly to the specified DMA channel. The operation of the DMA Mux in this mode is completely transparent to the system.

- Periodic Trigger Mode

  In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the Periodic Interrupt Timer (PIT).

DMA channels 0–3 may be used in all the modes listed above but channels 4–15 may be configured only to disabled or normal mode.

20.2 External signal description

20.2.1 Overview

The DMA Mux has no external pins.

20.3 Memory map and register definition

This section provides a detailed description of all memory-mapped registers in the DMA Mux.

20.3.1 Memory map

Table 243 shows the memory map for the DMA Mux. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA Mux.

Table 243. DMA_MUX memory map

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Location
0x0000	Channel #0 Configuration (CHCONFIG0)	*on page 508*
0x0001	Channel #1 Configuration (CHCONFIG1)	*on page 508*
0x0002	Channel #2 Configuration (CHCONFIG2)	*on page 508*
0x0003	Channel #3 Configuration (CHCONFIG3)	*on page 508*
0x0004	Channel #4 Configuration (CHCONFIG4)	*on page 508*

Table 243. DMA_MUX memory map (continued)

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Location
0x0005	Channel #5 Configuration (CHCONFIG5)	*on page 508*
0x0006	Channel #6 Configuration (CHCONFIG6)	*on page 508*
0x0007	Channel #7 Configuration (CHCONFIG7)	*on page 508*
0x0008	Channel #8 Configuration (CHCONFIG8)	*on page 508*
0x0009	Channel #9 Configuration (CHCONFIG9)	*on page 508*
0x000A	Channel #10 Configuration (CHCONFIG10)	*on page 508*
0x000B	Channel #11 Configuration (CHCONFIG11)	*on page 508*
0x000C	Channel #12 Configuration (CHCONFIG12)	*on page 508*
0x000D	Channel #13 Configuration (CHCONFIG13)	*on page 508*
0x000E	Channel #14 Configuration (CHCONFIG14)	*on page 508*
0x000F	Channel #15 Configuration (CHCONFIG15)	*on page 508*
0x0010	Channel #16 Configuration (CHCONFIG16)	*on page 508*
0x0011	Channel #17 Configuration (CHCONFIG17)	*on page 508*
0x0012	Channel #18 Configuration (CHCONFIG18)	*on page 508*
0x0013	Channel #19 Configuration (CHCONFIG19)	*on page 508*
0x0014	Channel #20 Configuration (CHCONFIG20)	*on page 508*
0x0015	Channel #21 Configuration (CHCONFIG21)	*on page 508*
0x0016	Channel #22 Configuration (CHCONFIG22)	*on page 508*
0x0017	Channel #23 Configuration (CHCONFIG23)	*on page 508*
0x0018	Channel #24 Configuration (CHCONFIG24)	*on page 508*
0x0019	Channel #25 Configuration (CHCONFIG25)	*on page 508*
0x001A	Channel #26 Configuration (CHCONFIG26)	*on page 508*
0x001B	Channel #27 Configuration (CHCONFIG27)	*on page 508*
0x001C	Channel #28 Configuration (CHCONFIG28)	*on page 508*
0x001D	Channel #29 Configuration (CHCONFIG29)	*on page 508*
0x001E	Channel #30 Configuration (CHCONFIG30)	*on page 508*
0x001F–0x3FFF	Reserved	

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit READ/WRITE to address ‘Base + 0x0000’, but performing a 32-bit access to address ‘Base + 0x0001’ is illegal.

20.3.2 Register descriptions

Channel Configuration Registers

Each of the DMA channels can be independently enabled/disabled and associated with one of the #SRC + #ALE total DMA sources in the system.

Figure 239. Channel Configuration Registers (CHCONFIG#*n*)

Address: Base + #*n* Access: User read/write

	7	6	5	4	3	2	1	0
R W	ENBL	TRIG	SOURCE					
Reset	0	0	0	0	0	0	0	0

Table 244. CHCONFIG#*x* field descriptions

Field	Description
7 ENBL	DMA Channel Enable ENBL enables the DMA Channel. 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA Mux. The DMA has separate channel enables/disables that should be used to disable or reconfigure a DMA channel. 1 DMA channel is enabled.
6 TRIG	DMA Channel Trigger Enable (for triggered channels only) TRIG enables the periodic trigger capability for the DMA Channel. 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the DMA Channel routes the specified source to the DMA channel. 1 Triggering is enabled.
5–0 SOURCE	DMA Channel Source (slot) SOURCE specifies which DMA source, if any, is routed to a particular DMA channel. See *Table 246*.

Table 245. Channel and trigger enabling

ENBL	TRIG	Function	Mode
0	X	DMA Channel is disabled	Disabled Mode
1	0	DMA Channel is enabled with no triggering (transparent)	Normal Mode
1	1	DMA Channel is enabled with triggering	Periodic Trigger Mode

Note: Setting multiple CHCONFIG registers with the same Source value will result in unpredictable behavior.

Note: Before changing the trigger or source settings a DMA channel must be disabled via the CHCONFIG[#n].ENBL bit.

20.4 DMA request mapping

Table 246. DMA channel mapping

DMA_CH_MUX channel	Module	DMA requesting module	DMA Mux input #
1	DSPI_0	DSPI_0 TX	DMA MUX Source #1
2	DSPI_0	DSPI_0 RX	DMA MUX Source #2
3	DSPI_1	DSPI_1 TX	DMA MUX Source #3
4	DSPI_1	DSPI_1 RX	DMA MUX Source #4
5	DSPI_2	DSPI_2 TX	DMA MUX Source #5
6	DSPI_2	DSPI_2 RX	DMA MUX Source #6
7	DSPI_3	DSPI_3 TX	DMA MUX Source #7
8	DSPI_3	DSPI_3 RX	DMA MUX Source #8
9	CTU_0	CTU	DMA MUX Source #9
10	CTU_0	CTU FIFO 1	DMA MUX Source #10
11	CTU_0	CTU FIFO 2	DMA MUX Source #11
12	CTU_0	CTU FIFO 3	DMA MUX Source #12
13	CTU_0	CTU FIFO 4	DMA MUX Source #13
14	Not connected		DMA MUX Source #14
15	DSPI_4	DSPI_4 TX	DMA MUX Source #15
16	etimer_0	eTimer_0 CH0	DMA MUX Source #16
17	etimer_0	eTimer_0 CH1	DMA MUX Source #17
18	etimer_1	eTimer_1 CH0	DMA MUX Source #18
19	etimer_1	eTimer_1 CH1	DMA MUX Source #19
20	adc_0	ADC_0	DMA MUX Source #20
21	DSPI_4	DSPI_4 RX	DMA MUX Source #21
22	—	ALWAYS requestors	DMA MUX Source #22
23	—	ALWAYS requestors	DMA MUX Source #23
24	—	ALWAYS requestors	DMA MUX Source #24
25	—	ALWAYS requestors	DMA MUX Source #25
26	—	ALWAYS requestors	DMA MUX Source #26
27	—	ALWAYS requestors	DMA MUX Source #27
28	—	ALWAYS requestors	DMA MUX Source #28
29	—	ALWAYS requestors	DMA MUX Source #29
30	—	ALWAYS requestors	DMA MUX Source #30

20.5 Functional description

This section provides a complete functional description of the DMA Mux. The primary purpose of the DMA Mux is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA Mux is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in *Section 20.6.2: Enabling and configuring sources* is followed, the configuration of the DMA Mux may be changed during the normal operation of the system.

Functionally, the DMA Mux channels may be divided into two classes: Channels that implement the normal routing functionality plus periodic triggering capability, and channels that implement only the normal routing functionality.

20.5.1 DMA channels with periodic triggering capability

Besides the normal routing functionality, the first four channels of the DMA Mux provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the Periodic Interrupt Timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to *Chapter 32: Periodic Interrupt Timer (PIT)* for more information on this topic.

Note: Because of the dynamic nature of the system (i.e., DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

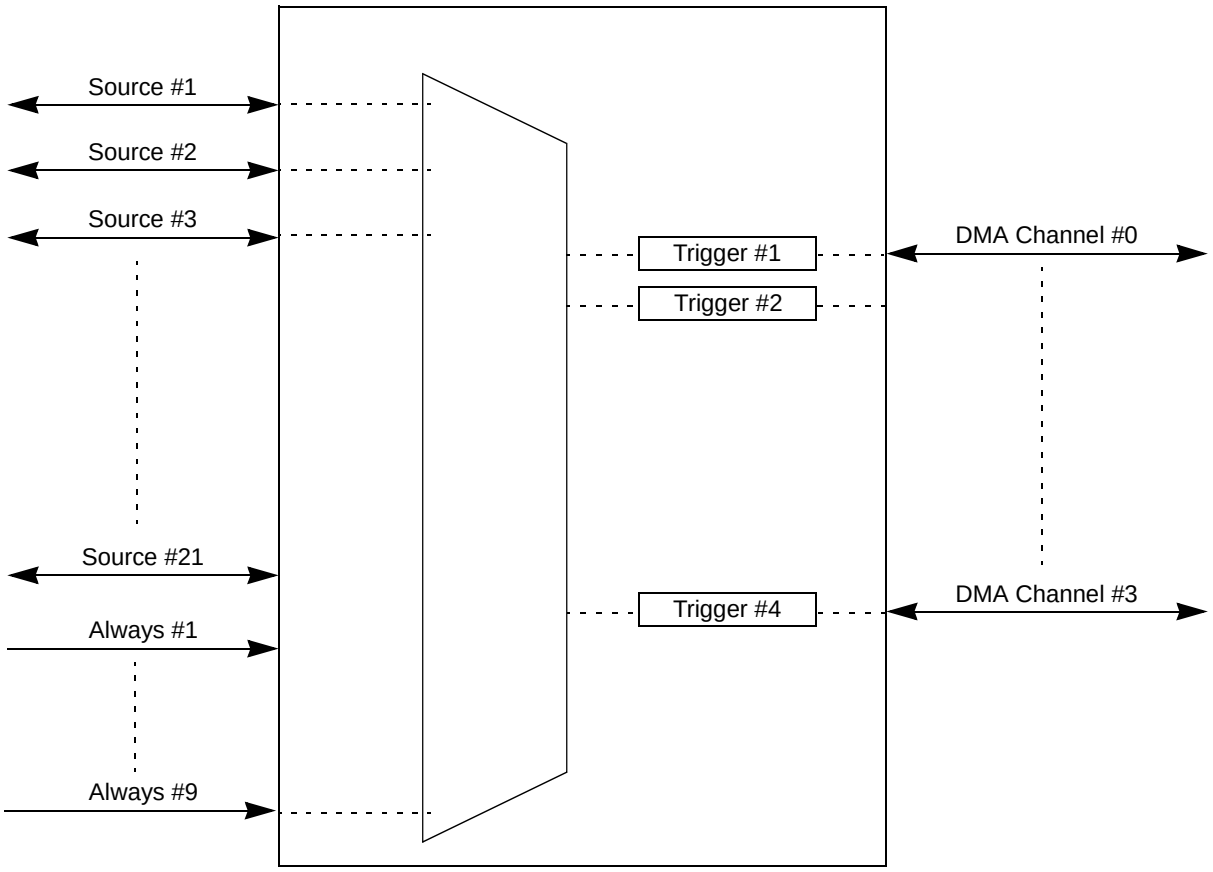


Figure 240. DMA mux triggered channels diagram

The DMA channel triggering capability allows the system to “schedule” regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the DMA until a trigger event has been seen. This is illustrated in *Figure 241*.

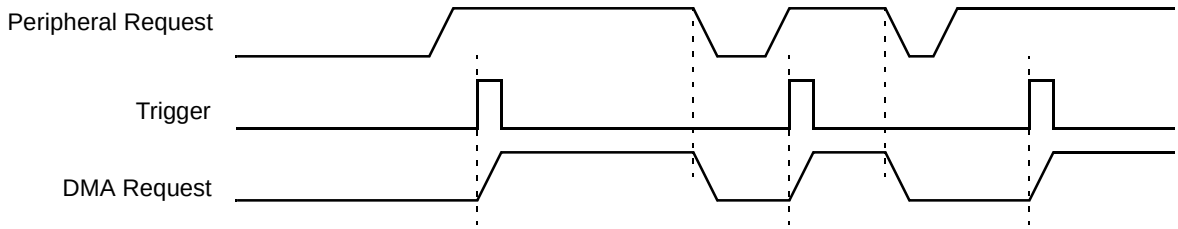


Figure 241. DMA mux channel triggering: normal operation

Once the DMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in *Figure 242*.

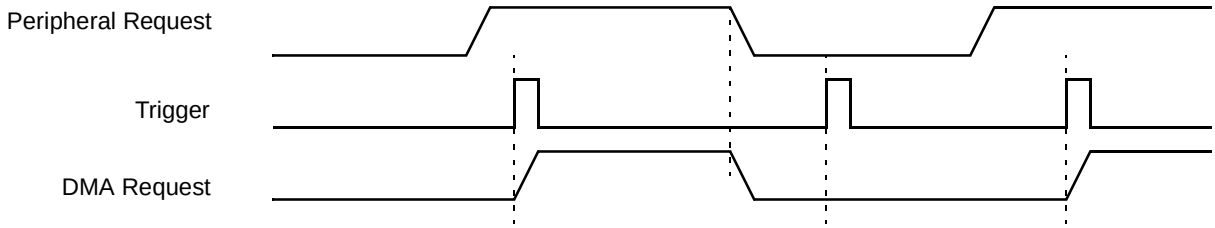


Figure 242. DMA mux channel triggering: ignored trigger

This triggering capability may be used with any peripheral that supports DMA transfers, and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. Once setup, the SPI will request DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5 µs (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO Ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (for example, resolution, range of values, etc.) may be found in *Chapter 32: Periodic Interrupt Timer (PIT)*.

20.5.2 DMA channels with no triggering capability

Channels 4–15 of the DMA Mux provide the normal routing functionality as described in *Section 20.1.3: Modes of operation.*

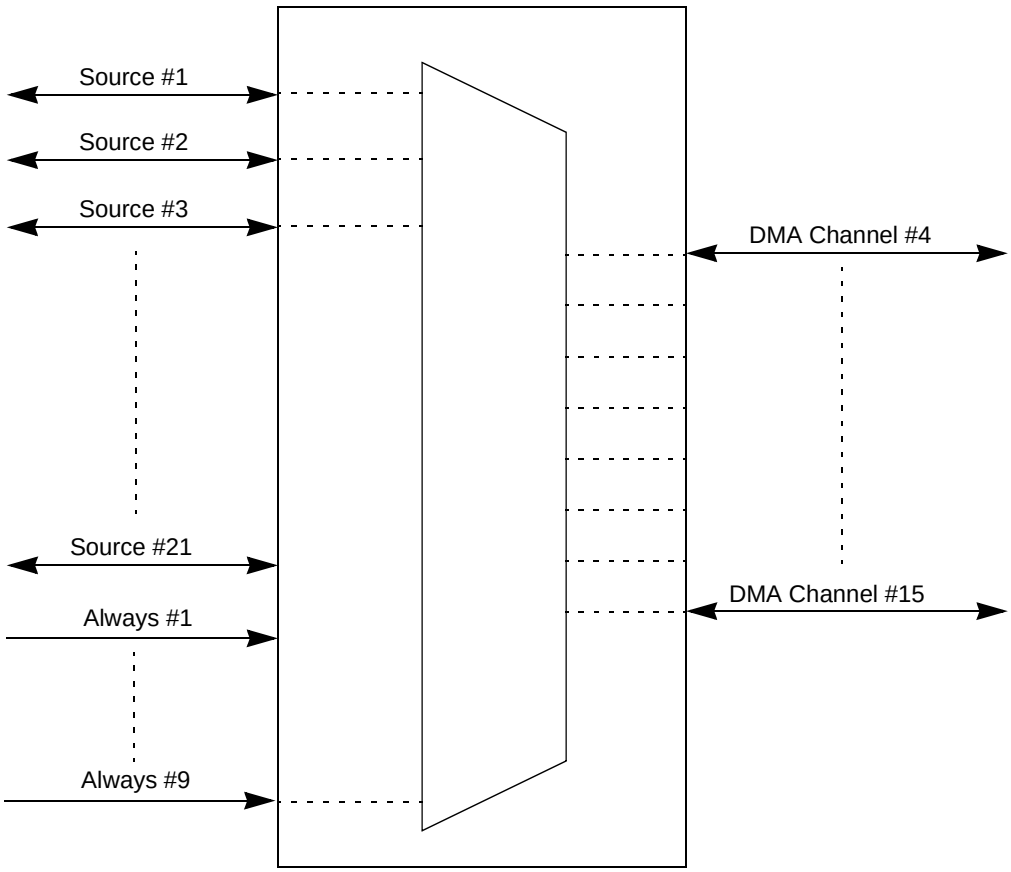


Figure 243. DMA mux channel 4–15 block diagram

20.6 Initialization/application information

20.6.1 Reset

The reset state of each individual bit is shown within the register description section (see *Section 20.3.2: Register descriptions*). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

20.6.2 Enabling and configuring sources

Enabling a source with periodic triggering:

1. Determine with which DMA channel the source will be associated. Remember that only the first four DMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. In the PIT, configure the corresponding timer.
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 8 Configure source #5 Transmit for use with DMA channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02).
2. Configure Channel 2 in the DMA, including enabling the channel.
3. Configure Timer 3 in the Periodic Interrupt Timer (PIT) for the desired trigger interval.
4. Write 0xC5 to CHCONFIG2 (Base Address + 0x02).

The following code example illustrates steps 1 and 4 above:

```
In File registers.h:
     #define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
     /* Following example assumes char is 8-bits */
     volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
     volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
     volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
     volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
     volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
     volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
     volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
     volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
     volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
     volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
     volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
     volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
     volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
     volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
     volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
     volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
```

```
#include "registers.h"
      :
      :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

Enabling a source without periodic triggering:

1. Determine with which DMA channel the source will be associated. Remember that only DMA channels 0–7 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

Example 9 Configure source #5 Transmit for use with DMA Channel 2, with no periodic triggering capability.

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02).
2. Configure Channel 2 in the DMA, including enabling the channel.
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02).

The following code example illustrates steps 1 and 3 above:

```
In File registers.h:
      #define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
      /* Following example assumes char is 8-bits */
      volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
      volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
      volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
      volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
      volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
      volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
      volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
      volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
      volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
      volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
      volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
      volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
      volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
      volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
      volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
```

```
     volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
     #include "registers.h"
          :
          :
     *CHCONFIG2 = 0x00;
     *CHCONFIG2 = 0x85;
```

Disabling a source:

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

Switching the source of a DMA channel:

1. Disable the DMA channel in the DMA and reconfigure the channel for the new source.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 10 Switch DMA Channel 8 from source #5 transmit to source #7 transmit

1. In the DMA configuration registers, disable DMA channel 8 and reconfigure it to handle the DSPI_0 transmits. This example assumes channel 0..7 have triggering capability (#TRG = 8).
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08).
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08). In this case, setting the TRIG bit would have no effect, because channels 8 and above do not support the periodic triggering functionality.

The following code example illustrates steps 2 and 3 above:

```
In File registers.h:
     #define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
     /* Following example assumes char is 8-bits */
     volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
     volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
     volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
     volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
     volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
     volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
     volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
     volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
     volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
     volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
```

```
     volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
     volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
     volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
     volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
     volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
     volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);
```

In File **main.c:**

```
     #include "registers.h"
          :
          :
     *CHCONFIG8 = 0x00;
     *CHCONFIG8 = 0x87;
```

21 FlexRay Communication Controller (FlexRay)

21.1 Introduction

21.1.1 Color coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

21.1.2 Overview

The controller is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The controller has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the controller with its surrounding modules is given in *Figure 244*.

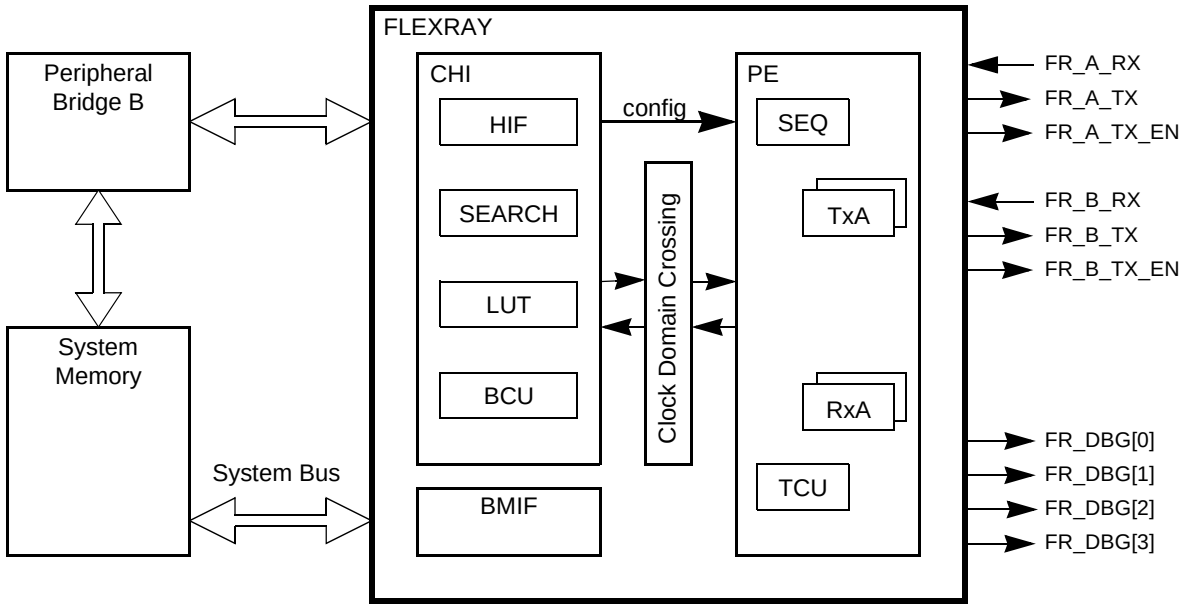


Figure 244. FlexRay block diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control

unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the FlexRay memory.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The controller stores the frame header and payload data of frames received or of frames to be transmitted in the FlexRay memory. The application accesses the FlexRay memory to retrieve and provide the frames to be processed by the controller. In addition to the frame header and payload data, the controller stores the synchronization frame related tables in the FlexRay memory for application processing.

The FlexRay memory is located in the system memory of the MCU. The controller has access to the FlexRay memory via its bus master interface (BMIF). The host provides the start address of the FlexRay memory window within the system memory by programming the *System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)*. All FlexRay memory related offsets are stored in offset registers. The physical address pointer into the FlexRay memory window of the MCU system memory is calculated using the offset values the FlexRay memory base address.

Note: The controller does not provide a memory protection scheme for the FlexRay memory.

21.1.3 Features

The controller provides the following features:

- FlexRay Communications System Protocol Specification, Version 2.1 Rev A compliant protocol implementation
- FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A compliant bus driver interface
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 64 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory
  - Allows for flexible and efficient message buffer implementation
  - Consistent data access ensured by means of buffer locking scheme
  - Application can lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0 to 254 bytes
- 2 independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
  - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as
  - Receive message buffer
  - Single buffered transmit message buffer
  - Double buffered transmit message buffer (combines two single buffered message buffer)
- Individual message buffer reconfiguration supported
  - Means provided to safely disable individual message buffers
  - Disabled message buffers can be reconfigured
- 2 independent receive FIFOs
  - 1 receive FIFO per channel
  - As many as 255 entries for each FIFO
  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for the dynamic segment
- 4 configurable slot error counters
- 4 dedicated slot status indicators
  - Used to observe slots without using receive message buffers

- Measured value indicators for the clock synchronization
  - Internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay memory
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- 1 absolute timer
- 1 timer that can be configured to absolute or relative

21.1.4 Modes of operation

This section describes the basic operational power modes of the controller.

Disabled mode

The controller enters the Disabled Mode during hard reset. The controller indicates that it is in the Disabled Mode by negating the module enable bit MEN in the *Module Configuration Register (MCR)*.

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in *Section 21.5.2: Register descriptions.*

The application configures the controller by accessing the configuration bits and fields in the *Module Configuration Register (MCR)*

Leave disabled mode

The controller leaves the Disabled Mode and enters the Normal Mode, when the application writes 1 to the module enable bit MEN in the *Module Configuration Register (MCR)*.

Note: When the controller was enabled, it cannot be disabled the later on.

Normal mode

In this mode the controller is fully functional. The controller indicates that it is in Normal Mode by asserting the module enable bit MEN in the *Module Configuration Register (MCR)*.

Enter normal mode

This mode is entered when the application requests the controller to leave the Disabled Mode. If the Normal Mode was entered by leaving the Disabled Mode, the application has to perform the protocol initialization described in *Section : Protocol initialization.* to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the *Module Configuration Register (MCR)*, the corresponding FlexRay bus driver ports are enabled and driven.

21.2 External signal description

This section lists and describes the controller signals, connected to external pins. These signals are summarized in *Table 247* and described in detail in *Section 21.2.1: Detailed signal descriptions.*

Note: The off-chip signals FR_A_RX, FR_A_TX, and FR_A_TX_EN are available on each package option. The availability of the other off-chip signals depends on the package option.

Table 247. External signal properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
FR_A_TX_EN	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
FR_B_TX_EN	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

21.2.1 Detailed signal descriptions

This section provides a detailed description of the controller signals, connected to external pins.

FR_A_RX — receive data channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

FR_A_TX — transmit data channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

FR_A_TX_EN — transmit enable channel A

The FR_A_TX_EN signal indicates to the FlexRay bus driver that the controller is attempting to transmit data on channel A.

FR_B_RX — receive data channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

FR_B_TX — transmit data channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

FR_B_TX_EN — transmit enable channel B

The FR_B_TX_EN signal indicates to the FlexRay bus driver that the controller is attempting to transmit data on channel B.

FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0] — strobe signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to *Section 21.6.16: Strobe signal support.*

21.3 Controller host interface clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. Since the FlexRay protocol requires data delivery at fixed points in time, the memory read cycles from the FlexRay memory must be finished after a fixed amount of time. To ensure this, a minimum frequency f_{chi} of the CHI clock is required, which is given in *Equation 21*.

Equation 21 $f_{chi} \geq 32MHz$

Additional requirements for the minimum frequency of the CHI clock result from the number of message buffers. These requirements are provided in *Section 21.7.3: Number of usable message buffers.*

21.4 Protocol engine clocking

The clock for the protocol engine can be generated by two sources. The first source is the internal crystal oscillator and the second source is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the *Module Configuration Register (MCR)*.

21.4.1 PLL Clocking

If the protocol engine is clocked by the internal FMPLL_0, select FVCO/6 from FMPLL_0 to have 80MHz. In this case the system clock is available with the constraint of 60 MHz.

21.4.2 Oscillator Clocking

If the protocol engine is clocked by the internal crystal oscillator, a 40Mhz crystal clock must be connected to the oscillator pins.

21.5 Memory map and register description

The controller occupies 512 bytes of address space starting at the controller's base address defined by the memory map of the MCU.

21.5.1 Memory map

The complete memory map of the controller is shown in *Table 248*. The addresses presented here are the offsets relative to the controller base address 0xFFFE_0000.

Table 248. FlexRay memory map

Offset	Register	Access
Module Configuration and Control		
0x0000	*Module Version Register (MVR)*	R
0x0002	*Module Configuration Register (MCR)*	R/W
0x0004	*System Memory Base Address High Register (SYMBADHR)*	R/W
0x0006	*System Memory Base Address Low Register (SYMBADLR)*	R/W
0x0008	*Strobe Signal Control Register (STBSCR)*	R/W
0x000A	Reserved	R
0x000C	*Message Buffer Data Size Register (MBDSR)*	R/W
0x000E	*Message Buffer Segment Size and Utilization Register (MBSSUTR)*	R/W
Test Registers		
0x0010	Reserved	R
0x0012	Reserved	R
Interrupt and Error Handling		
0x0014	*Protocol Operation Control Register (POCR)*	R/W
0x0016	*Global Interrupt Flag and Enable Register (GIFER)*	R/W
0x0018	*Protocol Interrupt Flag Register 0 (PIFR0)*	R/W
0x001A	*Protocol Interrupt Flag Register 1 (PIFR1)*	R/W
0x001C	*Protocol Interrupt Enable Register 0 (PIER0)*	R/W
0x001E	*Protocol Interrupt Enable Register 1 (PIER1)*	R/W
0x0020	*CHI Error Flag Register (CHIERFR)*	R/W
0x0022	*Message Buffer Interrupt Vector Register (MBIVEC)*	R
0x0024	*Channel A Status Error Counter Register (CASERCR)*	R
0x0026	*Channel B Status Error Counter Register (CBSERCR)*	R
Protocol Status		
0x0028	*Protocol Status Register 0 (PSR0)*	R
0x002A	*Protocol Status Register 1 (PSR1)*	R
0x002C	*Protocol Status Register 2 (PSR2)*	R
0x002E	*Protocol Status Register 3 (PSR3)*	R/W
0x0030	*Macrotick Counter Register (MTCTR)*	R
0x0032	*Cycle Counter Register (CYCTR)*	R
0x0034	*Slot Counter Channel A Register (SLTCTAR)*	R
0x0036	*Slot Counter Channel B Register (SLTCTBR)*	R
0x0038	*Rate Correction Value Register (RTCORVR)*	R
0x003A	*Offset Correction Value Register (OFCORVR)*	R
0x003C	*Combined Interrupt Flag Register (CIFRR)*	R
0x003E	*System Memory Access Time-Out Register (SYMATOR)*	R/W
Sync Frame Counter and Tables		
0x0040	*Sync Frame Counter Register (SFCNTR)*	R
0x0042	*Sync Frame Table Offset Register (SFTOR)*	R/W
0x0044	*Sync Frame Table Configuration, Control, Status Register (SFTCCSR)*	R/W

Table 248. FlexRay memory map (continued)

Offset	Register	Access
Sync Frame Filter		
0x0046	*Sync Frame ID Rejection Filter Register (SFIDRFR)*	R/W
0x0048	*Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)*	R/W
0x004A	*Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)*	R/W
Network Management Vector		
0x004C	*Network Management Vector Register 0 (NMVR0)*	R
0x004E	*Network Management Vector Register 1 (NMVR1)*	R
0x0050	*Network Management Vector Register 2 (NMVR2)*	R
0x0052	*Network Management Vector Register 3 (NMVR3)*	R
0x0054	*Network Management Vector Register 4 (NMVR4)*	R
0x0056	*Network Management Vector Register 5 (NMVR5)*	R
0x0058	*Network Management Vector Length Register (NMVLR)*	R/W
Timer Configuration		
0x005A	*Timer Configuration and Control Register (TICCR)*	R/W
0x005C	*Timer 1 Cycle Set Register (TI1CYSR)*	R/W
0x005E	*Timer 1 Macrotick Offset Register (TI1MTOR)*	R/W
0x0060	*Timer 2 Configuration Register 0 (TI2CR0)*	R/W
0x0062	*Timer 2 Configuration Register 1 (TI2CR1)*	R/W
Slot Status Configuration		
0x0064	*Slot Status Selection Register (SSSR)*	R/W
0x0066	*Slot Status Counter Condition Register (SSCCR)*	R/W
Slot Status		
0x0068	*Slot Status Register 0 (SSR0)*	R
0x006A	*Slot Status Register 1 (SSR1)*	R
0x006C	*Slot Status Register 2 (SSR2)*	R
0x006E	*Slot Status Register 3 (SSR3)*	R
0x0070	*Slot Status Register 4 (SSR4)*	R
0x0072	*Slot Status Register 5 (SSR5)*	R
0x0074	*Slot Status Register 6 (SSR6)*	R
0x0076	*Slot Status Register 7 (SSR7)*	R
0x0078	*Slot Status Counter Register 0 (SSCR0)*	R
0x007A	*Slot Status Counter Register 1 (SSCR1)*	R
0x007C	*Slot Status Counter Register 2 (SSCR2)*	R
0x007E	*Slot Status Counter Register 3 (SSCR3)*	R
MTS Generation		
0x0080	*MTS A Configuration Register (MTSACFR)*	R/W
0x0082	*MTS B Configuration Register (MTSBCFR)*	R/W
Shadow Buffer Configuration		
0x0084	*Receive Shadow Buffer Index Register (RSBIR)*	R/W
Receive FIFO — Configuration		

Table 248. FlexRay memory map (continued)

Offset	Register	Access
0x0086	*Receive FIFO Selection Register (RFSR)*	R/W
0x0088	*Receive FIFO Start Index Register (RFSIR)*	R/W
0x008A	*Receive FIFO Depth and Size Register (RFDSR)*	R/W
Receive FIFO - Status		
0x008C	*Receive FIFO A Read Index Register (RFARIR)*	R
0x008E	*Receive FIFO B Read Index Register (RFBRIR)*	R
Receive FIFO - Filter		
0x0090	*Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)*	R/W
0x0092	*Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)*	R/W
0x0094	*Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)*	R/W
0x0096	*Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)*	R/W
0x0098	*Receive FIFO Range Filter Configuration Register (RFRFCFR)*	R/W
0x009A	*Receive FIFO Range Filter Control Register (RFRFCTR)*	R/W
Dynamic Segment Status		
0x009C	*Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)*	R
0x009E	*Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)*	R
Protocol Configuration		
0x00A0 ... 0x00DC	*Protocol Configuration Register 0 (PCR0)* ... *Protocol Configuration Register 30 (PCR30)*	R/W ... R/W
0x00DE ... 0x00FE	Reserved	R
Message Buffers Configuration, Control, Status		
0x0100	*Message Buffer Configuration, Control, Status Register 0 (MBCCSR0)*	R/W
0x0102	*Message Buffer Cycle Counter Filter Register 0 (MBCCFR0)*	R/W
0x0104	*Message Buffer Frame ID Register 0 (MBFIDR0)*	R/W
0x0106	*Message Buffer Index Register 0 (MBIDXR0)*	R/W
...	...	...
0x02F8	*Message Buffer Configuration, Control, Status Register 63 (MBCCSR63)*	R/W
0x02FA	*Message Buffer Cycle Counter Filter Register 63 (MBCCFR63)*	R/W
0x02FC	*Message Buffer Frame ID Register 63 (MBFIDR63)*	R/W
0x02FE	*Message Buffer Index Register 63 (MBIDXR63)*	R/W

21.5.2 Register descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 249 provides a key for the register figures and register tables.

Table 249. Register access conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
–	Not defined after reset and not affected by reset.

Register reset

All registers except the *Message Buffer Cycle Counter Filter Registers (MBCCFRn)*, *Message Buffer Frame ID Registers (MBFIDRn)*, and *Message Buffer Index Registers (MBIDXRn)* are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in *Table 250*.

Table 250. Additional register reset conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command “0101” to the POCCMD field in the *Protocol Operation Control Register (POCR)*.
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit MBCCSRn[EDT] while the message buffer is enabled (MBCCSn[EDS] = 1) and the controller grants the disable to the application by clearing the MBCCSRn[EDS] bit.

Register write access

This section describes the write access restriction terms that apply to all registers.

Register write access restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in *Table 251*. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.

Table 251. Register write access restrictions

Condition	Indication	Description
Any Time	—	No write access restriction.
Disabled Mode	MCR[MEN] = 0	Write access only when the controller is in Disabled Mode.
Normal Mode	MCR[MEN] = 1	Write access only when the controller is in Normal Mode.
POC:config	PSR0[PROTSTATE] = *POC:config*	Write access only when the Protocol is in the *POC:config* state.
MB_DIS	MBCCSR[EDS] = 0	Write access only when the related Message Buffer is disabled.
MB_LCK	MBCCSRn[LCKS] = 1	Write access only when the related Message Buffer is locked.

Register write access requirements

All registers can be accessed with 8-bit, 16-bit and 32-bit wide operations. For some of the registers, at least a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected

Internal register access

The following memory mapped registers are used to access multiple internal registers.

- *Strobe Signal Control Register (STBSCR)*
- *Slot Status Selection Register (SSSR)*
- *Slot Status Counter Condition Register (SSCCR)*
- *Receive Shadow Buffer Index Register (RSBIR)*

Each of these memory mapped registers provides a SEL field and a WMD bit. The SEL field selects the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

Module Version Register (MVR)

Base + 0x0000

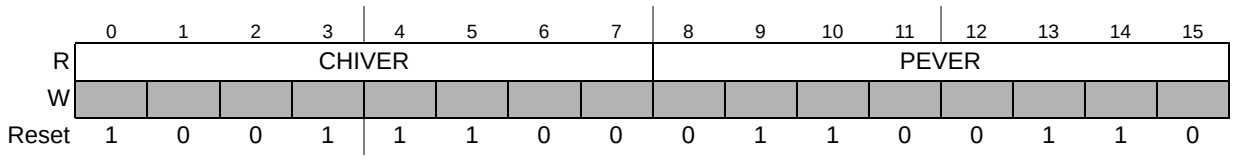


Figure 245. Module Version Register (MVR)

This register provides the controller version number. The module version number is derived from the CHI version number and the PE version number.

Table 252. MVR field descriptions

Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the controller host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

Module Configuration Register (MCR)

Base + 0x0002

Write: MEN, SBFF, SCM, CHB, CHA, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	0	R*	0	0	0	CLK SEL	BITRATE			0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 246. Module Configuration Register (MCR)

This register defines the global configuration of the controller.

Table 253. MCR field descriptions

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the controller is in the Disabled Mode. The application requests the controller to leave the Disabled Mode by writing 1 to this bit Before leaving the Disabled Mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, BITRATE values. For details see *Section 21.1.4: Modes of operation.* 0 Write: ignored, controller disable not possible. Read: controller disabled. 1 Write: enable controller. Read: controller enabled. If the controller is enabled it cannot be disabled.
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the controller in case of a system bus failure. 0 Continue normal operation. 1 Transition to freeze mode.
SCM	**Single Channel Device Mode** — This control bit defines the channel device mode of the controller as described in *Section 21.6.10: Channel device modes.* 0 controller works in dual channel device mode. 1 controller works in single channel device mode.
CHB CHA	**Channel Enable** — protocol related parameter: *pChannels* The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given *Table 254*.

Table 253. MCR field descriptions

Field	Description
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see *Section 21.6.15: Sync frame filtering.* 0 Synchronization frame filtering disabled. 1 Synchronization frame filtering enabled.
R*	**Reserved** — This bit is reserved. It is read as 0. Application must not write 1 to this bit.
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator. 1 PE clock source is generated by on-chip PLL.
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the bit rate of the FlexRay channels according to *Table 255*.

Table 254. FlexRay channel selection

SCM	CHB	CHA	Description
Dual channel device modes			
0	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by controller ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by controller
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by controller
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by controller ports FR_B_RX, FR_B_TX, and FR_A_TX_EN driven by controller - connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN driven by controller - connected to FlexRay channel B
Single channel device mode			
1	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by controller ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by controller
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by controller
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by controller - connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by controller
	1	1	reserved

Table 255. FlexRay channel bit rate selection

MCR[BITRATE]	FlexRay channel bit rate [Mbit/s]
000	10.0
001	5.0
010	2.5
011	8.0
100	reserved
101	reserved
110	reserved
111	reserved

System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)

Base + 0x0004 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SYS_MEM_BASE_ADDR[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 247. System Memory Base Address High Register (SYMBADHR)

Base + 0x0006 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SYS_MEM_BASE_ADDR[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 248. System Memory Base Address Low Register (SYMBADLR)

Note: The system memory base address must be set before the controller is enabled.

The system memory base address registers define the base address of the FlexRay memory within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 256. SYMBADHR and SYMBADLR field descriptions

Field	Description
SYS_MEM_BASE_ADDR	This base address will be added to all system memory offset values stored in registers or calculated in the controller before the controller accesses the system memory via its bus master interface. The system memory base address must be aligned to an 16-byte boundary.

Strobe Signal Control Register (STBSCR)

Base + 0x0008 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SEL							0	0	0	ENB	0	0	STBPSEL	
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 249. Strobe Signal Control Register (STBSCR)

This register assigns the individual protocol timing related strobe signals given in *Table 258* to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to *Section 21.6.16: Strobe signal support.*

Note: In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 257. STBSCR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in *Table 258* to be enabled or disabled and assigned to one of the four strobe ports given in *Table 258*.
ENB	**Strobe Signal Enable** — This control bit enables and disables the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 Assign selected signal to FR_DBG[0]. 01 Assign selected signal to FR_DBG[1]. 10 Assign selected signal to FR_DBG[2]. 11 Assign selected signal to FR_DBG[3].

Table 258. Strobe signal mapping

SEL		Description	Channel	Type	Offset(1)	Reference
dec	hex					
0	0x00	poc_startup_state[0] (for coding see PSR0[4])	—	value	0	MT start
1	0x01	poc_startup_state[1] (for coding see PSR0[5])				
2	0x02	poc_startup_state[2] (for coding see PSR0[6])				
3	0x03	poc_startup_state[3] (for coding see PSR0[7])				
4	0x04	poc_state[0] (for coding see PSR0[8])				
5	0x05	poc_state[1] (for coding see PSR0[9])				
6	0x06	poc_state[2] (for coding see PSR0[10])				
7	0x07	channel idle indicator	A	level	+5	FR_A_RX
8	0x08		B			FR_B_RX
9	0x09	receive data after glitch filtering	A	value	+4	FR_A_RX
10	0x0A		B			FR_B_RX
11	0x0B	synchronization edge strobe	A	pulse	+4	FR_A_RX
12	0x0C		B			FR_B_RX
13	0x0D	header received	A	pulse	+4	FR_A_RX
14	0x0E		B			FR_B_RX
15	0x0F	wakeup symbol decoded	A	pulse	+5	FR_A_RX
16	0x10		B			FR_B_RX
17	0x11	MTS or CAS symbol decoded	A	pulse	+4	FR_A_RX
18	0x12		B			FR_B_RX
19	0x13	frame decoded	A	pulse	+4	FR_A_RX
20	0x14		B			FR_B_RX
21	0x15	channel idle detected	A	pulse	+4	FR_A_RX
22	0x16		B			FR_B_RX
23	0x17	start of communication element detected	A	pulse	+4	FR_A_RX
24	0x18		B			FR_B_RX
25	0x19	potential frame start channel	A	pulse	+4	FR_A_RX
26	0x1A		B			FR_B_RX
27	0x1B	wakeup collision detected	A	pulse	+5	FR_A_RX
28	0x1C		B			FR_B_RX
29	0x1D	content error detected	A	level	+4	FR_A_RX
30	0x1E		B			FR_B_RX
31	0x1F	syntax error detected	A	pulse	+4	FR_A_RX
32	0x20		B			FR_B_RX
33	0x21	start transmission of wakeup pattern	A	pulse	-1	FR_A_TX
34	0x22		B			FR_B_TX
35	0x23	start transmission of MTS or CAS symbol	A	pulse	-1	FR_A_TX
36	0x24		B			FR_B_TX
37	0x25	start of transmission	A	pulse	-1	FR_A_TX
38	0x26		B			FR_B_TX
39	0x27	end of transmission	A	pulse	-1	FR_A_TX
40	0x28		B			FR_B_TX

Table 258. Strobe signal mapping (continued)

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
41	0x29	static segment indicator	—	level	0	MT start
42	0x2A	dynamic segment indicator	—	level	0	MT start
43	0x2B	symbol window indicator	—	level	0	MT start
44	0x2C	NIT indicator	—	level	0	MT start
45	0x2D	action point	—	pulse	-1	FR_A_TX
46	0x2E	sync calculation complete[2]	—	pulse	—	—
47	0x2F	start of offset correction	—	pulse	-2	MT start
48	0x30	cycle count[0]	—	value	-2	MT start
49	0x31	cycle count[1]				
50	0x32	cycle count[2]				
51	0x33	cycle count[3]				
52	0x34	cycle count[4]				
53	0x35	cycle count[5]				
54	0x36	slot count[0]	A	value	0	MT start
55	0x37	slot count[1]				
56	0x38	slot count[2]				
57	0x39	slot count[3]				
58	0x3A	slot count[4]				
59	0x3B	slot count[5]				
60	0x3C	slot count[6]				
61	0x3D	slot count[7]				
62	0x3E	slot count[8]				
63	0x3F	slot count[9]				
64	0x40	slot count[10]				
65	0x41	slot count[0]	B	value	0	MT start
66	0x42	slot count[1]				
67	0x43	slot count[2]				
68	0x44	slot count[3]				
69	0x45	slot count[4]				
70	0x46	slot count[5]				
71	0x47	slot count[6]				
72	0x48	slot count[7]				
73	0x49	slot count[8]				
74	0x4A	slot count[9]				
75	0x4B	slot count[10]				
76	0x4C	cycle start	—	pulse	0	MT start
77	0x4D	slot start	A	pulse	0	MT start
78	0x4E		B			
79	0x4F	minislot start	—	pulse	0	MT start
80	0x50	arm	—	value	+1	MT start
81	0x51	mt	—	value	+1	MT start

1. Given in PE clock cycles
2. Indicates internal PE event not directly related to FlexRay bus timing

Message Buffer Data Size Register (MBDSR)

Base + 0x000C Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MBSEG2DS							0	MBSEG1DS						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 250. Message Buffer Data Size Register (MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The controller provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 259. MBDSR field descriptions

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

Message Buffer Segment Size and Utilization Register (MBSSUTR)

Base + 0x000E Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	LAST_MB_SEG1						0	0	LAST_MB_UTIL					
W																
Reset	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1

Figure 251. Message Buffer Segment Size and Utilization Register (MBSSUTR)

This register defines the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 260. MBSSUTR field descriptions

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers MBCCSRn, MBCCFRn, MBFIDRn, MBIDXRn with n <= LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG1+1 individual message buffers. The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers MBCCSRn, MBCCFRn, MBFIDRn, MBIDXRn with LAST_MB_SEG1 < n < 32. If LAST_MB_SEG1 = 31 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number n <= LAST_MB_UTIL. If LAST_MB_UTIL=LAST_MB_SEG1 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

Protocol Operation Control Register (POCR)

Base + 0x0014 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 252. Protocol Operation Control Register (POCR)

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see *Section 21.7.4: Protocol control command execution.*

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to *Section 21.6.11: External clock synchronization.*

Table 261. POCR field descriptions

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.
EOC_AP	**External Offset Correction Application** — This field triggers the application of the external offset correction value defined in the *Protocol Configuration Register 29 (PCR29)*. 00 Do not apply external offset correction value. 01 Reserved. 10 Subtract external offset correction value. 11 Add external offset correction value.
ERC_AP	**External Rate Correction Application** — This field triggers application of the external rate correction value defined in the *Protocol Configuration Register 21 (PCR21)* 00 Do not apply external rate correction value. 01 Reserved. 10 Subtract external rate correction value. 11 Add external rate correction value.
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The controller sets this status bit when the application has issued a protocol control command via the POCCMD field. The controller clears this status bit when protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the controller ignores this command, sets the protocol command ignored error flag PCMI_EF in the *CHI Error Flag Register (CHIERFR)*, and will not change the value of the POCCMD field. 0Command write idle, command accepted and ready to receive new protocol command. 1Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The controller sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed(1) transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 Reserved. 1010 Reserved. 1011 Reserved. 1100 RESET(2) — Immediately reset the Protocol Engine (see *Section 21.7.5: Protocol reset command*). 1101 Reserved. 1110 Reserved. 1111 Reserved.

1. Delayed means on completion of current communication cycle.
2. Additional to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*

Global Interrupt Flag and Enable Register (GIFER)

Base + 0x0016 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUPIF	FNEBIF	FNEAIF	RBIF	TBIF	MIE	PRIE	CHIE	WUPIE	FNEBIE	FNEAIE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 253. Global Interrupt Flag and Enable Register (GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is shown in *Figure 388*. For more details on interrupt generation, see *Section 21.6.20: Interrupt support.* These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 262. GIFER field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if at least one of the other interrupt flags is in this register is asserted and the related interrupt enable is asserted, too. The controller generates the module interrupt request if MIE is asserted. 0 No interrupt flag is asserted or no interrupt enable is set. 1 At least one of the other interrupt flags in this register is asserted and the related interrupt bit is asserted, too.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Protocol Interrupt Flag Register 0 (PIFR0)* and *Protocol Interrupt Flag Register 1 (PIFR1)* is asserted and the related interrupt enable flag is asserted, too. The controller generates the combined protocol interrupt request if the PRIE flag is asserted. 0 All individual protocol interrupt flags are equal to 0 or no interrupt enable bit is set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *CHI Error Flag Register (CHIERFR)* is asserted and the chi error interrupt enable GIFER[CHIE] is asserted. The controller generates the combined CHI error interrupt if the CHIE flag is asserted, too. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled. 1 At least one CHI error flag is asserted and chi error interrupt is enabled.
WUPIF	**Wakeup Interrupt Flag** — This flag is set when the controller has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the *Protocol Status Register 3 (PSR3).* The controller generates the wakeup interrupt request if the WUPIE flag is asserted. 0 No wakeup condition or interrupt disabled. 1 Wakeup symbol received on FlexRay bus and interrupt enabled.

Table 262. GIFER field descriptions (continued)

Field	Description
FNEBIF	**Receive FIFO channel B Not Empty Interrupt Flag** — This flag is set when the receive FIFO for channel B is not empty. If the application writes 1 to this bit, the controller updates the FIFO status, increments or wraps the FIFO read index in the *Receive FIFO B Read Index Register (RFBRIR)* and clears the interrupt flag if the FIFO B is now empty. If the FIFO is still not empty, the controller sets this flag again. The controller generates the Receive FIFO B Not empty interrupt if the FNEBIE flag is asserted. 0 Receive FIFO B is empty or interrupt is disabled. 1 Receive FIFO B is not empty and interrupt enabled.
FNEAIF	**Receive FIFO channel A Not Empty Interrupt Flag** — This flag is set when the receive FIFO for channel A is not empty. If the application writes 1 to this bit, the controller updates the FIFO status, increments or wraps the FIFO read index in the *Receive FIFO A Read Index Register (RFARIR)* and clears the interrupt flag if the FIFO A is now empty. If the FIFO is still not empty, the controller sets this flag again. The controller generates the Receive FIFO A Not empty interrupt if the FNEAIE flag is asserted. 0 Receive FIFO A is empty or interrupt is disabled. 1 Receive FIFO A is not empty and interrupt enabled.
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (MBCCSn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Message Buffer Configuration, Control, Status Registers (MBCCSRn)* are asserted. The application cannot clear this RBIF flag directly. This flag is cleared by the controller when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag asserted. 1 At least one individual receive message buffer has the MBIF and MBIE flag asserted.
TBIF	**Transmit Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Message Buffer Configuration, Control, Status Registers (MBCCSRn)* are equal to 1. The application cannot clear this TBIF flag directly. This flag is cleared by the controller when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the host has cleared the interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag asserted. 1 At least one individual transmit message buffer has the MBIF and MBIE flag asserted.
MIE	**Module Interrupt Enable** — This flag controls if the module interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	**Protocol Interrupt Enable** — This flag controls if the protocol interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	**CHI Interrupt Enable** — This flag controls if the CHI interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
WUPIE	**Wakeup Interrupt Enable** — This flag controls if the wakeup interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Table 262. GIFER field descriptions (continued)

Field	Description
FNEBIE	**Receive FIFO channel B Not Empty Interrupt Enable** — This flag controls if the receive FIFO B interrupt line is asserted when the FNEBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FNEAIE	**Receive FIFO channel A Not Empty Interrupt Enable** — This flag controls if the receive FIFO A interrupt line is asserted when the FNEAIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
RBIE	**Receive Buffer Interrupt Enable** — This flag controls if the receive buffer interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
TBIE	**Transmit Interrupt Enable** — This flag controls if the transmit buffer interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Protocol Interrupt Flag Register 0 (PIFR0)

Base + 0x0018 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 254. Protocol Interrupt Flag Register 0 (PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 263. PIFR0 field description

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: 1) *pLatestTx* violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the *Protocol Configuration Register 14 (PCR14)* and *Protocol Configuration Register 15 (PCR15)* when the CONFIG_COMPLETE command was sent by the application via the *Protocol Operation Control Register (POCR)*. If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the *Protocol Configuration Register 3 (PCR3)*. 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event 1 Insufficient number of measurements for rate correction detected
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the *offset_coorection_out* field in the *Protocol Configuration Register 9 (PCR9)* and the *rate_correction_out* field in the *Protocol Configuration Register 14 (PCR14)*. 0 No such event. 1 Offset or rate correction limit reached.

Table 263. PIFR0 field description (continued)

Field	Description
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the *Protocol Configuration Register 30 (PCR30)*. 0 No such event. 1 More than *node_sync_max* sync frames detected. Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

Protocol Interrupt Flag Register 1 (PIFR1)

Base + 0x001A Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 255. Protocol Interrupt Flag Register 1 (PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 264. PIFR1 field descriptions

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the *Protocol Status Register 0 (PSR0)* is changed by the controller. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the *Protocol Operation Control Register (POCR)*, and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see *Section 21.7.4: Protocol control command execution.* 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the controller has detected a communication failure between the protocol engine and the controller host interface 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the *Protocol Status Register 0 (PSR0)* has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding *Slot Status Counter Registers (SSCR0–SSCR3)* is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the controller has written the sync frame measurement / ID tables into the FlexRay memory for the even cycle. 0 No such event. 1 Sync frame measurement table written
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the controller has written the sync frame measurement / ID tables into the FlexRay memory for the odd cycle. 0 No such event. 1 Sync frame measurement table written

Protocol Interrupt Enable Register 0 (PIER0)

Base + 0x001C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 256. Protocol Interrupt Enable Register 0 (PIER0)

This register defines whether or not the individual interrupt flags defined in the *Protocol Interrupt Flag Register 0 (PIFR0)* can generate a protocol interrupt request.

Table 265. PIER0 field descriptions

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.

Table 265. PIER0 field descriptions

Field	Description
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
TBVB_IE	**Transmission across boundary on channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
TBVA_IE	**Transmission across boundary on channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0interrupt request generation disabled. 1interrupt request generation enabled.

Protocol Interrupt Enable Register 1 (PIER1)

Base + 0x001E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PEC F_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 257. Protocol Interrupt Enable Register 1 (PIER1)

This register defines whether or not the individual interrupt flags defined in *Protocol Interrupt Flag Register 1 (PIFR1)* can generate a protocol interrupt request.

Table 266. PIER1 field descriptions

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0interrupt request generation disabled. 1 interrupt request generation enabled.
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

CHI Error Flag Register (CHIERFR)

Base + 0x0020 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	DBL_EF	SBCF_EF	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 258. CHI Error Flag Register (CHIERFR)

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the *Global Interrupt Flag and Enable Register (GIFER)*.

Table 267. CHIERFR field descriptions

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event 1 Frame lost on channel B detected
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error 1 Frame lost on channel A detected
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the *Protocol Operation Control Register (POCR)* while the BSY flag is equal to 1. In this case the command is ignored by the controller and is lost. 0 No such error 1 POC command ignored
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the Receive FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 Receive FIFO overrun on channel B has been detected
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the Receive FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 Receive FIFO overrun on channel B has been detected
MSB_EF	**Message Buffer Search Error Flag** — This flag is set if the message buffer search engine is still running while the next search cycle must be started due to the FlexRay protocol timing. In this case, not all message buffers are considered while searching. 0 No such event 1 Search engine active while search start appears
MBU_EF	Message Buffer Utilization Error Flag — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the *Message Buffer Segment Size and Utilization Register (MBSSUTR)*. If the application writes to a MBCCSRn register with n > LAST_MB_UTIL, the controller ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the *CHI Error Flag Register (CHIERFR)*. 0 No such event 1 Non-utilized message buffer enabled.
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the controller due to internal operations. In that case, the controller does not grant the lock to the application. The application must issue the lock request again. 0 No such error 1 Lock error detected

Table 267. CHIERFR field descriptions (continued)

Field	Description
DBL_EF	**Double Transmit Message Buffer Lock Error Flag** — This flag is set if the application tries to lock the transmit side of a double transmit message buffer. In this case, the controller does not grant the lock to the transmit side of a double transmit message buffer. 0 No such event 1 Double transmit buffer lock error occurred
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see *Section : System bus access timeout*). 0 No such event 1 System bus access not finished in time
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred 1 Frame ID error occurred
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the *Protocol Configuration Register 24 (PCR24)*. 0 No such error occurred 1 Dynamic payload length error occurred
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the *Protocol Configuration Register 19 (PCR19)*. 0 No such error occurred 1 Static payload length error occurred
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the *Network Management Vector Length Register (NMVLR)*. In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred 1 Network management length error occurred
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see *Network Management Vector Registers (NMVR0–NMVR5)*) are not updated. 0 No such error occurred 1 Network management frame error occurred
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the controller (see *Section : System bus illegal address access*). 0 No such event 1 Illegal system bus address accessed

Message Buffer Interrupt Vector Register (MBIVEC)

Base + 0x0022

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	TBIVEC						0	0	RBIVEC					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 259. Message Buffer Interrupt Vector Register (MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 268. MBIVEC field descriptions

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

Channel A Status Error Counter Register (CASERCR)

Base + 0x0024 **A**dditional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 260. Channel A Status Error Counter Register (CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The controller increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see *Section 21.6.18: Slot status monitoring.*

Table 269. CASERCR field descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

Channel B Status Error Counter Register (CBSERCR)

Base + 0x0026 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 261. Channel B Status Error Counter Register (CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The controller increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see *Section 21.6.18: Slot status monitoring.*

Table 270. CBSERCR field descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

Protocol Status Register 0 (PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 262. Protocol Status Register 0 (PSR0)

This register provides information about the current protocol status.

Table 271. PSR0 field descriptions

Field	Description
ERRMODE	**Error Mode** — protocol related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 reserved
SLOTMODE	**Slot Mode** — protocol related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 reserved
PROTSTATE	**Protocol State** — protocol related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*
STARTUP STATE	**Startup State** — protocol related variable: *vPOC!StartupState.* This field indicates the current sub-state of the startup procedure. 0000 reserved 0001 reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integration listen* 0110 reserved 0111 *POC:initialize schedule* 1000 reserved 1001 reserved 1010 *POC:coldstart consistency check* 1011 reserved 1100 reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — protocol related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 reserved

Protocol Status Register 1 (PSR1)

Base + 0x002A Additional Reset: CSAA, CSP, CPN: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 263. Protocol Status Register 1 (PSR1)

Table 272. PSR1 field descriptions

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — protocol related event: 'set coldstart abort indicator in CHI' This flag is set when the controller has aborted a cold start attempt. 0 No such event 1 Cold start attempt aborted
CSP	**Leading Cold Start Path** — This status bit is set when the controller has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event 1 *POC:normal active* reached from *POC:startup* state via leading cold start path
REMCSAT	**Remaining Coldstart Attempts** — protocol related variable: *vRemainingColdstartAttempts* This field provides the number of remaining cold start attempts that the controller will execute.
CPN	**Leading Cold Start Path Noise** — protocol related variable: *vPOC!ColdstartNoise* This status bit is set if the controller has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the controller was starting up the cluster. 0 No such event 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path
HHR	**Host Halt Request Pending** — protocol related variable: *vPOC!CHIHaltRequest* This status bit is set when controller receives the HALT command from the application via the *Protocol Operation Control Register (POCR)*. The controller clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 HALT command received
FRZ	**Freeze Occurred** — protocol related variable: *vPOC!Freeze* This status bit is set when the controller has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The controller clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 Immediate halt due to FREEZE or internal error condition
APTAC	**Allow Passive to Active Counter** — protocol related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the *Protocol Configuration Register 12 (PCR12)*.

Protocol Status Register 2 (PSR2)

Base + 0x002C Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 264. Protocol Status Register 2 (PSR2)

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the controller after the end of the NIT and before the end of the first slot of the next communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the controller after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the controller after the end of the static segment and before the end of the current communication cycle.

Table 273. PSR2 field descriptions

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEB	**NIT Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event 1 Syntax error detected
STCB	**Symbol Window Transmit Conflict on Channel B** — protocol related variable: *vSS!TxConflict* for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event 1 Transmission conflict detected
SBVB	**Symbol Window Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEB	**Symbol Window Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event 1 Syntax error detected

Table 273. PSR2 field descriptions (continued)

Field	Description
MTB	**Media Access Test Symbol MTS Received on Channel B** — protocol related variable: *vSS!ValidMTS* for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event 1 MTS symbol received
NBVA	**NIT Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEA	**NIT Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event 1 Syntax error detected
STCA	**Symbol Window Transmit Conflict on Channel A** — protocol related variable: *vSS!TxConflict* for symbol window on channel A This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event 1 Transmission conflict detected
SBVA	**Symbol Window Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEA	**Symbol Window Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event 1 Syntax error detected
MTA	**Media Access Test Symbol MTS Received on Channel A** — protocol related variable: *vSS!ValidMTS* for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received 0 No such event
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — protocol related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the *Protocol Configuration Register 8 (PCR8)*. The controller resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

Protocol Status Register 3 (PSR3)

Base + 0x002E Additional Reset: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 265. Protocol Status Register 3 (PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 274. PSR3 field descriptions

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received 1 Wakeup symbol received
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received 0 No syntactically valid frames received
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received 1 Wakeup symbol received

Table 274. PSR3 field descriptions (continued)

Field	Description
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received 1 At least one syntactically valid frame received

Macrotick Counter Register (MTCTR)

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MTCT													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 266. Macrotick Counter Register (MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 275. MTCTR field descriptions

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

Cycle Counter Register (CYCTR)

Base + 0x0032

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	CYCCNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 267. Cycle Counter Register (CYCTR)

This register provides the number of the current communication cycle.

Table 276. CYCTR field descriptions

Field	Description
CYCCNT	**Cycle Counter** — protocol related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

Slot Counter Channel A Register (SLTCTAR)

Base + 0x0034

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 268. Slot Counter Channel A Register (SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 277. SLTCTAR field descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — protocol related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

Slot Counter Channel B Register (SLTCTBR)

Base + 0x0036

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 269. Slot Counter Channel B Register (SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 278. SLTCTBR field descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel B** — protocol related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

Rate Correction Value Register (RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RATECORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 270. Rate Correction Value Register (RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The controller updates this register during the NIT of each odd numbered communication cycle.

Table 279. RTCORVR field descriptions

Field	Description
RATECORR	**Rate Correction Value** — protocol related variable: *vRateCorrection* (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the *Protocol Configuration Register 13 (PCR13)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Protocol Interrupt Flag Register 0 (PIFR0)*. If the controller was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

Offset Correction Value Register (OFCORVR)

Base + 0x003A Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OFFSETCORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 271. Offset Correction Value Register (OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The controller updates this register during the NIT.

Table 280. OFCORVR field descriptions

Field	Description
OFFSET-CORR	**Offset Correction Value** — protocol related variable: *vOffsetCorrection* (before value limitation and external offset correction) This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the *Protocol Configuration Register 29 (PCR29)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Protocol Interrupt Flag Register 0 (PIFR0)*. If the controller was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

Combined Interrupt Flag Register (CIFRR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUPIF	FNEBIF	FNEAIF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 272. Combined Interrupt Flag Register (CIFRR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is shown in *Figure 389*. The individual interrupt flags WUPIF, FNEBIF, and FNEAIF are copies of corresponding flags in the *Global Interrupt Flag and Enable Register (GIFER)* and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the *Global Interrupt Flag and Enable Register (GIFER)*.

Note: The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Global Interrupt Flag and Enable Register (GIFER).

Table 281. CIFRR field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted. 1 At least one interrupt source has its interrupt flag asserted.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Protocol Interrupt Flag Register 0 (PIFR0)* or *Protocol Interrupt Flag Register 1 (PIFR1)* is equal to 1. 0 All individual protocol interrupt flags are equal to 0. 1 At least one of the individual protocol interrupt flags is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *CHI Error Flag Register (CHIERFR)* is equal to 1. 0 All CHI error flags are equal to 0. 1 At least one CHI error flag is equal to 1.
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as GIFER[WUPIF]
FNEBIF	**Receive FIFO channel B Not Empty Interrupt Flag** — Provides the same value as GIFER[FNEBI]
FNEAIF	**Receive FIFO channel A Not Empty Interrupt Flag** — Provides the same value as GIFER[FNEAIF]
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (MBCCSRn[MTD] = 0) the interrupt flag MBIF in the corresponding *Message Buffer Configuration, Control, Status Registers (MBCCSRn)* is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSRn[MTD] = 1) the interrupt flag MBIF in the corresponding *Message Buffer Configuration, Control, Status Registers (MBCCSRn)* is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

System Memory Access Time-Out Register (SYMATOR)

Base + 0x003E Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	TIMEOUT				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 273. System Memory Access Time-Out Register (SYMATOR)

Table 282. SYMATOR field descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines the maximum amount of time to finish a system bus access in order to ensure correct frame transmission and reception (see *Section : System bus access timeout*).

Sync Frame Counter Register (SFCNTR)

Base + 0x0040 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFEVB				SFEVA				SFODB				SFODA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 274. Sync Frame Counter Register (SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

Note: If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the controller will not update the fields SFEVB and SFEVA.

If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the controller will not update the values SFODB and SFODA.

Table 283. SFCNTR field descriptions

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vsSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	Sync Frames Channel A, even cycle — protocol related variable: size of (*vsSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, odd cycle — protocol related variable: size of (*vsSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, odd cycle — protocol related variable: size of (*vsSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

Sync Frame Table Offset Register (SFTOR)

Base + 0x0042 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFT_OFFSET[15:1]															0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 275. Sync Frame Table Offset Register (SFTOR)

This register defines the FlexRay Memory related offset for sync frame tables. For more details, see *Section 21.6.12: Sync frame ID and sync frame deviation tables.*

Table 284. SFTOR field description

Field	Description
SFTOR	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the FlexRay Memory. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

Base + 0x0044 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM						ELKS	OLKS	EVAL	OVAL	0	0	SDVEN	SIDEN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 276. Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see *Section 21.6.12: Sync frame ID and sync frame deviation tables.*

Table 285. SFTCCSR field descriptions

Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit locks and unlocks the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit locks and unlocks the odd cycle tables. 0 No effect 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The controller clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).

Table 285. SFTCCSR field descriptions

Field	Description
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The controller clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OPT	**One Pair Trigger** — This trigger bit controls whether the controller writes continuously or only one pair of Sync Frame Tables into the FlexRay memory. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the controller writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the FlexRay memory. In this case, the controller clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the controller writes continuously the enabled Sync Frame Tables into the FlexRay memory. 0 Write continuously pairs of enabled Sync Frame Tables into FlexRay memory. 1 Write only one pair of enabled Sync Frame Tables into FlexRay memory.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the controller to write the Sync Frame Deviation Tables into the FlexRay memory. 0 Do not write Sync Frame Deviation Tables. 1 Write Sync Frame Deviation Tables into FlexRay memory. If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the controller to write the Sync Frame ID Tables into the FlexRay memory. 0 Do not write Sync Frame ID Tables. 1 Write Sync Frame ID Tables into FlexRay memory.

Sync Frame ID Rejection Filter Register (SFIDRFR)

Base + 0x0046 16-bit write access required Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SYNFRID									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 277. Sync Frame ID Rejection Filter Register (SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the controller accepts the sync frame in the current cycle.

Table 286. SFIDRFR field descriptions

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see *Section : Sync frame rejection filtering.*

Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

Base + 0x0048 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FVAL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 278. Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see *Section 21.6.15: Sync frame filtering.*

Table 287. SFIDAFVR field descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

Base + 0x004A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FMSK									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 279. Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see *Section : Sync frame acceptance filtering.*

Table 288. SFIDAFMR field descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

Network Management Vector Registers (NMVR0–NMVR5)

Base + 0x004C (NMVR0)
Base + 0x004E (NMVR1)
Base + 0x0050 (NMVR2)
Base + 0x0052 (NMVR3)
Base + 0x0054 (NMVR4)
Base + 0x0056 (NMVR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMVP[15:8]								NMVP[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 280. Network Management Vector Registers (NMVR0–NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the *Network Management Vector Length Register (NMVLR)*. If NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the *Network Management Vector Registers (NMVR0–NMVR5)*, which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 289. NMVR[0:5] field descriptions

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the *Network Management Vector Registers (NMVR0–NMVR5)* and the receive message buffer payload bytes in NMV[0:11] is shown in ***Table 290***.

Table 290. Mapping of NMVRn to received payload bytes NMVn

NMVRn Register	NMVn Received Payload
NMVR0[NMVP[15:8]]	NMV0
NMVR0[NMVP[7:0]]	NMV1
NMVR1[NMVP[15:8]]	NMV2
NMVR1[NMVP[7:0]]	NMV3
...	
NMVR5[NMVP[15:8]]	NMV10
NMVR5[NMVP[7:0]]	NMV11

Network Management Vector Length Register (NMVLR)

Base + 0x0058 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 281. Network Management Vector Length Register (NMVLR)

This register defines the length of the network management vector in bytes.

Table 291. NMVLR field descriptions

Field	Description
NMVL	**Network Management Vector Length** — protocol related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

Timer Configuration and Control Register (TICCR)

Base + 0x005A Write: T2_CFG: *POC:config*
T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_	T2_	0	0	0	T2ST	0	0	0	T1_	0	0	0	T1ST
W			CFG	REP		T2SP	T2TR					REP		T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 282. Timer Configuration and Control Register (TICCR)

This register configures and controls the two timers T1 and T2. For timer details, see *Section 21.6.17: Timer support.* The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 292. TICCR field descriptions

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the time base mode of Timer T2. 0 T2 is absolute timer. 1 T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 T2 is non repetitive 1 T2 is repetitive
T2SP	**Timer T2 Stop** — This trigger bit stops timer T2. 0 no effect 1 stop timer T2
T2TR	**Timer T2 Trigger** — This trigger bit starts timer T2. 0 no effect 1 start timer T2

Table 292. TICCR field descriptions

Field	Description
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 timer T2 is idle 1 timer T2 is running
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 T1 is non repetitive 1 T1 is repetitive
T1SP	**Timer T1 Stop** — This trigger bit stops timer T1. 0 no effect 1 stop timer T1
T1TR	**Timer T1 Trigger** — This trigger bit starts timer T1. 0 no effect 1 start timer T1
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 timer T1 is idle 1 timer T1 is running

Note: Both timers are deactivated immediately when the protocol enters a state different from POC:normal active or POC:normal passive.

Timer 1 Cycle Set Register (TI1CYSR)

Base + 0x005C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 283. Timer 1 Cycle Set Register (TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to *Section : Absolute timer T1.*

Table 293. TI1CYSR field descriptions

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

Timer 1 Macrotick Offset Register (TI1MTOR)

Base + 0x005E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 284. Timer 1 Macrotick Offset Register (TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to *Section : Absolute timer T1.*

Table 294. TI1MTOR field descriptions

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

Timer 2 Configuration Register 0 (TI2CR0)

Base + 0x0060 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_CYC_VAL						0	0	T2_CYC_MSK					
W																
R	T2_MTCNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 285. Timer 2 Configuration Register 0 (TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the *Timer Configuration and Control Register (TICCR)*. For a detailed description of timer T2, refer to *Section : Absolute / relative timer T2.*

Table 295. TI2CR0 field descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

Timer 2 Configuration Register 1 (TI2CR1)

Base + 0x0062 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_MTOFFSET													
W																
R	T2_MTCNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 286. Timer 2 Configuration Register 1 (TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the *Timer Configuration and Control Register (TICCR)*. For a detailed description of timer T2, refer to *Section : Absolute / relative timer T2.*

Table 296. TI2CR1 field descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

Slot Status Selection Register (SSSR)

Base + 0x0064 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	SLOTNUMBER										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 287. Slot Status Selection Register (SSSR)

This register accesses the four internal non memory-mapped slot status selection registers SSSR0 to SSSR3. Each internal registers selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding *Slot Status Registers (SSR0–SSR7)* according to *Table 298*. For a detailed description of slot status monitoring, refer to *Section 21.6.18: Slot status monitoring.*

Table 297. SSSR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 select SSSR0. 01 select SSSR1. 10 select SSSR2. 11 select SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 298. Mapping between SSSRn and SSRn

Internal Slot Status Selection Register	Write the Slot Status of the Slot Selected by SSSRn for each			
	Even Communication Cycle		Odd Communication Cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
SSSR0	SSR0[15:8]	SSR0[7:0]	SSR1[15:8]	SSR1[7:0]
SSSR1	SSR2[15:8]	SSR2[7:0]	SSR3[15:8]	SSR3[7:0]
SSSR2	SSR4[15:8]	SSR4[7:0]	SSR5[15:8]	SSR5[7:0]
SSSR3	SSR6[15:8]	SSR6[7:0]	SSR7[15:8]	SSR7[7:0]

Slot Status Counter Condition Register (SSCCR)

Base + 0x0066 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 288. Slot Status Counter Condition Register (SSCCR)

This register accesses and programs the four internal non-memory mapped Slot Status Counter Condition Registers SSCCR0 to SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding *Slot Status Counter Registers (SSCR0–SSCR3)*. The correspondence is given in *Table 300*. For a detailed description of slot status counters,

refer to *Section : Slot status counter registers.*

Table 299. SSCCR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 select SSCCR0. 01 select SSCCR1. 10 select SSCCR2. 11 select SSCCR3.
CNTCFG	**Counter Configuration** — These bit field controls the channel related incrementing of the slot status counter. 00 increment by 1 if condition is fulfilled on channel A. 01 increment by 1 if condition is fulfilled on channel B. 10 increment by 1 if condition is fulfilled on at least one channel. 11 increment by 2 if condition is fulfilled on both channels channel. increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	**Valid Frame Restriction** — This bit restricts the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit restricts the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit restricts the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	**Startup Frame Restriction** — This bit restricts the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUS MASK[3:0]	**Slot Status Mask** — This bit field enables the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 300. Mapping between internal SSCCRn and SSCRn

Condition Register	Condition Defined for Register
SSCCR0	SSCR0
SSCCR1	SSCR1
SSCCR2	SSCR2
SSCCR3	SSCR3

Slot Status Registers (SSR0–SSR7)

Base + 0x0068 (SSR0)
Base + 0x006A (SSR1)
Base + 0x006C (SSR2)
Base + 0x006E (SSR3)
Base + 0x0070 (SSR4)
Base + 0x0072 (SSR5)
Base + 0x0074 (SSR6)
Base + 0x0076 (SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 289. Slot Status Registers (SSR0–SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the *Slot Status Selection Register (SSSR)*. Each register is updated after the end of the corresponding slot as shown in *Figure 386*. The register bits are directly related to the protocol variables and described in more detail in *Section 21.6.18: Slot status monitoring.*

Table 301. SSR0–SSR7 field descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.

Table 301. SSR0–SSR7 field descriptions

Field	Description
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

Slot Status Counter Registers (SSCR0–SSCR3)

Base + 0x0078 (SSCR0)
Base + 0x007A (SSCR1)
Base + 0x007C (SSCR2)
Base + 0x007E (SSCR3) Additional Reset: RUN Command

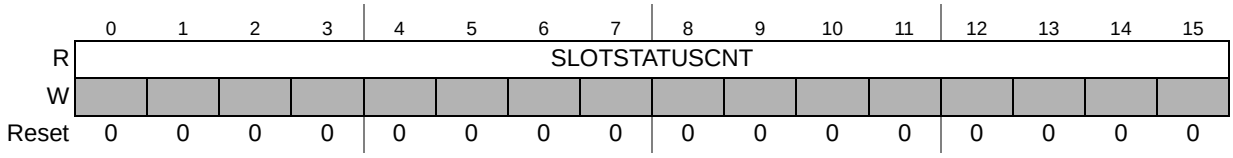


Figure 290. Slow Status Counter Registers (SSCR0–SSCR3)

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register SSCCRn, which can be programmed by using the *Slot Status Counter Condition Register (SSCCR)*. For more details, see *Section : Slot status counter registers.*

Note: If the counter has reached its maximum value 0xFFFF and is in the multicycle mode, i.e., SSCCRn[MCY] = 1, the counter is not reset to 0x0000. The application can reset the counter by clearing the SSCCRn[MCY] bit and waiting for the next cycle start, when the controller clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 302. SSCR0–SSCR3 field descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

MTS A Configuration Register (MTSACFR)

Base + 0x0080 Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

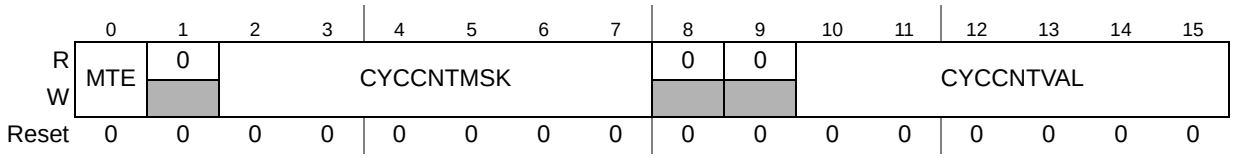


Figure 291. MTS A Configuration Register (MTSACFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see *Section 21.6.13: MTS generation.*

Table 303. MTSACFR field descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit enables and disables the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

MTS B Configuration Register (MTSBCFR)

Base + 0x0082

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

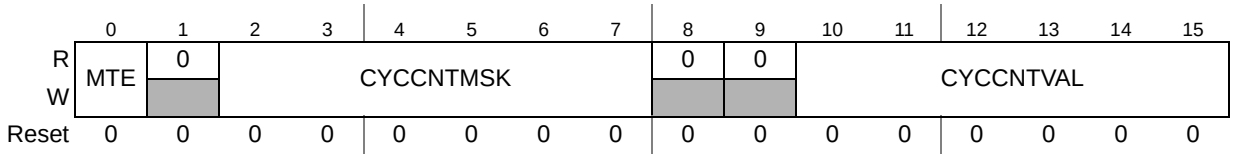


Figure 292. MTS B Configuration Register (MTSBCFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see *Section 21.6.13: MTS generation.*

Table 304. MTSBCFR field descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit enables and disables the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

Receive Shadow Buffer Index Register (RSBIR)

Base + 0x0084

16-bit write access required

Write: WMD, SEL: Any Time
RSBIDX: *POC:config*

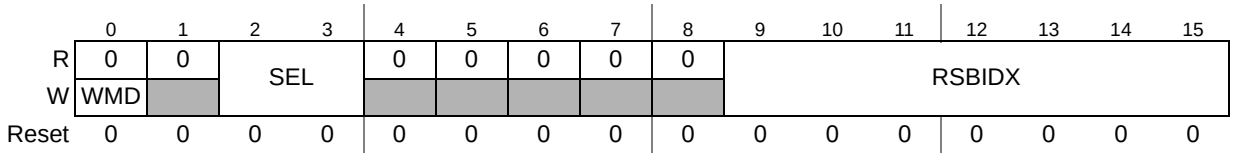


Figure 293. Receive Shadow Buffer Index Register (RSBIR)

This register provides and retrieves the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to *Section : Receive Shadow Buffers Concept.*

Table 305. RSBIR field descriptions

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	**Selector** — This field selects the internal receive shadow buffer index register for access. 00 RSBIR_A1 — receive shadow buffer index register for channel A, segment 1 01 RSBIR_A2 — receive shadow buffer index register for channel A, segment 2 10 RSBIR_B1 — receive shadow buffer index register for channel B, segment 1 11 RSBIR_B2 — receive shadow buffer index register for channel B, segment 2
RSBIDX	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The controller uses this index to determine the physical location of the shadow buffer header field in the FlexRay memory. The controller will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. controller: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index.

Receive FIFO Selection Register (RFSR)

Base + 0x0086 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SEL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 294. Receive FIFO Selection Register (RFSR)

This register selects a receiver FIFO for subsequent access through the receiver FIFO configuration registers summarized in *Table 306*.

Table 306. SEL controlled receiver FIFO registers

Register
Receive FIFO Start Index Register (RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (RFRFCFR)
Receive FIFO Range Filter Control Register (RFRFCTR)

Table 307. RFSR field descriptions

Field	Description
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected 1 Receiver FIFO for channel B selected

Receive FIFO Start Index Register (RFSIR)

Base + 0x0088 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 295. Receive FIFO Start Index Register (RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 308. RFSIR field descriptions

Field	Description
SIDX	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected receive FIFO. The controller uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FIFO_DEPTH								0	ENTRY_SIZE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 296. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, i.e., the number of entries and the size of each entry.

Table 309. RFDSR field descriptions

Field	Description
FIFO_DEPTH	**FIFO Depth** — This field defines the depth of the selected receive FIFO, i.e., the number of entries.
ENTRY_SIZE	**Entry Size** — This field defines the size of the frame data sections for the selected receive FIFO in 2 byte entities.

Receive FIFO A Read Index Register (RFARIR)

Base + 0x008C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 297. Receive FIFO A Read Index Register (RFARIR)

This register provides the message buffer header index of the next available receive FIFO A entry that the application can read.

Table 310. RFARIR field descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available receive FIFO message buffer that the application can read. The controller increments this index when the application writes to the FNEAIF flag in the *Global Interrupt Flag and Enable Register (GIFER)*. The index wraps back to the first message buffer header index if the end of the FIFO was reached.

Note: If the receive FIFO not empty flag FNEAIF is not set, the RDIDX field points to an physical message buffer which content is not valid. Only when FNEAIF is set, the message buffer indicated by RDIDX contains valid data.

Receive FIFO B Read Index Register (RFBRIR)

Base + 0x008E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 298. Receive FIFO B Read Index Register (RFBRIR)

This register provides the message buffer header index of the next available receive FIFO B entry that the application can read.

Table 311. RFBRIR field descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available receive FIFO entry that the application can read. The controller increments this index when the application writes to the FNEBIF flag in the *Global Interrupt Flag and Enable Register (GIFER)*.The index wraps back to the first message buffer header index if the end of the FIFO was reached.

Note: If the receive FIFO not empty flag FNEBIF is not set, the RDIDX field points to an physical message buffer which content is not valid. Only when FNEBIF is set, the message buffer indicated by RDIDX contains valid data.

Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

Base + 0x0090 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIDAFVAL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 299. Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

This register defines the filter value for the message ID acceptance filter of the selected receive FIFO. For details on message ID filtering see *Section : Receive FIFO filtering.*

Table 312. RFMIDAFVR field descriptions

Field	Description
MIDAFVAL	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

Base + 0x0092 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIDAFMSK															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 300. Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

This register defines the filter mask for the message ID acceptance filter of the selected receive FIFO. For details on message ID filtering see *Section : Receive FIFO filtering.*

Table 313. RFMIAFMR field descriptions

Field	Description
MIDAFMSK	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

Base + 0x0094 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFVAL										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 301. Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

This register defines the filter value for the frame ID rejection filter of the selected receive FIFO. For details on frame ID filtering see *Section : Receive FIFO filtering.*

Table 314. RFFIDRFVR field descriptions

Field	Description
FIDRFVAL	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFMSK										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 302. Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected receive FIFO. For details on frame ID filtering see *Section : Receive FIFO filtering.*

Table 315. RFFIDRFMR field descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

Receive FIFO Range Filter Configuration Register (RFRFCFR)

Base + 0x0098 16-bit write access required Write: WMD, IBD, SEL: Any Time

SID: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IBD	SEL		0	SID										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 303. Receive FIFO Range Filter Configuration Register (RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected receive FIFO. For details on frame ID range filter see *Section : Receive FIFO filtering.*

Table 316. RFRFCFR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 Program lower interval boundary. 1 Program upper interval boundary.
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 Select frame ID range filter 0. 01 Select frame ID range filter 1. 10 Select frame ID range filter 2. 11 Select frame ID range filter 3.
SID	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

Receive FIFO Range Filter Control Register (RFRFCTR)

Base + 0x009A Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 304. Receive FIFO Range Filter Control Register (RFRFCTR)

This register enables and disables each frame ID range filter and defines whether it is running as acceptance or rejection filter.

Table 317. RFRFCTR field descriptions

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 range filter 3 runs as acceptance filter. 1 range filter 3 runs as rejection filter.
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 range filter 2 runs as acceptance filter . 1 range filter 2 runs as rejection filter.
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 range filter 1 runs as acceptance filter. 1 range filter 1 runs as rejection filter.
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 range filter 0 runs as acceptance filter. 1 range filter 0 runs as rejection filter.

Table 317. RFRFCTR field descriptions (continued)

Field	Description
F3EN	**Range Filter 3 Enable** — This control bit enables and disables the frame ID range filter 3. 0 range filter 3 disabled. 1 range filter 3 enabled.
F2EN	**Range Filter 2 Enable** — This control bit enables and disables the frame ID range filter 2. 0 range filter 2 disabled. 1 range filter 2 enabled.
F1EN	**Range Filter 1 Enable** — This control bit enables and disables the frame ID range filter 1. 0 range filter 1 disabled. 1 range filter 1 enabled.
F0EN	**Range Filter 0 Enable** — This control bit enables and disables the frame ID range filter 0. 0 range filter 0 disabled. 1 range filter 0 enabled.

Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 305. Last Dynamic Slot Channel A Register (LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 318. LDTXSLAR field descriptions

Field	Description
LASTDYNTXSLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)

Base + 0x009E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 306. Last Dynamic Slot Channel B Register (LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 319. LDTXSLBR field descriptions

Field	Description
LASTDYNTXSLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

Protocol configuration registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in *Table 320*. For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Table 320. Protocol configuration register fields

Name	Description(1)	Min	Max	Unit	PCR
coldstart_attempts	gColdstartAttempts			number	3
action_point_offset	*gdActionPointOffset* - 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* - 1			gdBit	4
dynamic_slot_idle_phase	gdDynamicSlotIdlePhase			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* - 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* - 1			MT	2
static_slot_length	gdStaticSlot			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* - 1			MT	13
symbol_window_exists	*gdSymbolWindow*!=0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	gdTSSTransmitter			gdBit	5
wakeup_symbol_rx_idle	gdWakeupSymbolRxIdle			gdBit	5
wakeup_symbol_rx_low	gdWakeupSymbolRxLow			gdBit	3
wakeup_symbol_rx_window	gdWakeupSymbolRxWindow			gdBit	4
wakeup_symbol_tx_idle	gdWakeupSymbolTxIdle			gdBit	8
wakeup_symbol_tx_low	gdWakeupSymbolTxLow			gdBit	5
noise_listen_timeout	(*gListenNoise* × *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	pMacroInitialOffset[A]			MT	6
macro_initial_offset_b	pMacroInitialOffset[B]			MT	16
macro_per_cycle	gMacroPerCycle			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28

Table 320. Protocol configuration register fields (continued)

Name	Description[1]	Min	Max	Unit	PCR
max_without_clock_correction_fatal	gMaxWithoutClockCorrectionFatal			cyclepairs	8
max_without_clock_correction_passive	gMaxWithoutClockCorrectionPassive			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	gNumberOfStaticSlots			static slot	2
offset_correction_start	gOffsetCorrectionStart			MT	11
payload_length_static	gPayloadLengthStatic			2-bytes	19
max_payload_length_dynamic	pPayloadLengthDynMax			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	pAllowHaltDueToClock			bool	26
allow_passive_to_active	pAllowPassiveToActive			cyclepairs	12
cluster_drift_damping	pClusterDriftDamping			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	pKeySlotId			number	18
key_slot_used_for_startup	pKeySlotUsedForStartup			bool	11
key_slot_used_for_sync	pKeySlotUsedForSync			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	gSyncNodeMax			number	30
micro_initial_offset_a	pMicroInitialOffset[A]			µT	20
micro_initial_offset_b	pMicroInitialOffset[B]			µT	20
micro_per_cycle	pMicroPerCycle			µT	22/23
micro_per_cycle_min	pMicroPerCycle - pdMaxDrift			µT	24/25
micro_per_cycle_max	pMicroPerCycle + pdMaxDrift			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	pOffsetCorrectionOut			µT	9
rate_correction_out	pRateCorrectionOut			µT	14
single_slot_enabled	pSingleSlotEnabled			bool	10
wakeup_channel	pWakeupChannel	see *Table 321*			10
wakeup_pattern	pWakeupPattern			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	pExternOffsetCorrection			µT	29
extern_rate_correction	pExternRateCorrection			µT	21

1. See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 321. Wakeup channel selection

wakeup_channel	Wakeup Channel
0	A
1	B

Protocol Configuration Register 0 (PCR0)

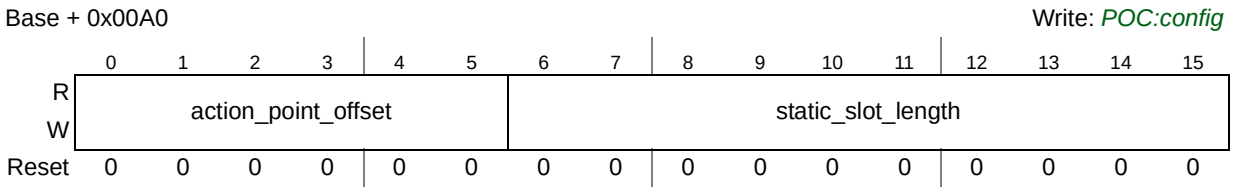


Figure 307. Protocol Configuration Register 0 (PCR0)

Protocol Configuration Register 1 (PCR1)

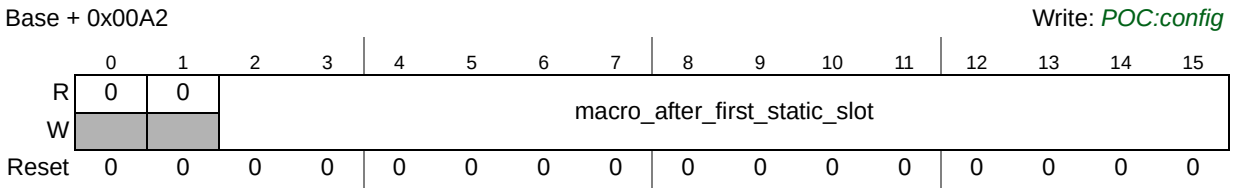


Figure 308. Protocol Configuration Register 1 (PCR1)

Protocol Configuration Register 2 (PCR2)

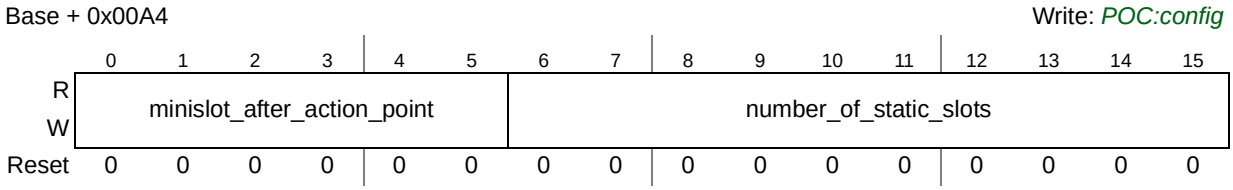


Figure 309. Protocol Configuration Register 2 (PCR2)

Protocol Configuration Register 3 (PCR3)

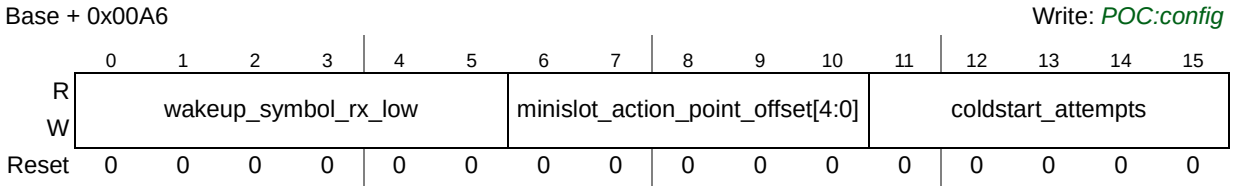


Figure 310. Protocol Configuration Register 3 (PCR3)

Protocol Configuration Register 4 (PCR4)

Base + 0x00A8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cas_rx_low_max								wakeup_symbol_rx_window							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 311. Protocol Configuration Register 4 (PCR4)

Protocol Configuration Register 5 (PCR5)

Base + 0x00AA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	tss_transmitter				wakeup_symbol_tx_low				wakeup_symbol_rx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 312. Protocol Configuration Register 5 (PCR5)

Protocol Configuration Register 6 (PCR6)

Base + 0x00AC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	symbol_window_after_action_point							macro_initial_offset_a							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 313. Protocol Configuration Register 6 (PCR6)

Protocol Configuration Register 7 (PCR7)

Base + 0x00AE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_b								micro_per_macro_nom_half							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 314. Protocol Configuration Register 7 (PCR7)

Protocol Configuration Register 8 (PCR8)

Base + 0x00B0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	max_without_clock_correction_fatal				max_without_clock_correction_passive				wakeup_symbol_tx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 315. Protocol Configuration Register 8 (PCR8)

Protocol Configuration Register 9 (PCR9)

Base + 0x00B2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	mini slot_ exists	sym bol_ win dow_ exists	offset_correction_out													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 316. Protocol Configuration Register 9 (PCR9)

Protocol Configuration Register 10 (PCR10)

Base + 0x00B4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	singl e_slo t _en abled	wake up_ chan nel	macro_per_cycle													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 317. Protocol Configuration Register 10 (PCR10)

Protocol Configuration Register 11 (PCR11)

Base + 0x00B6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	key_ slot_ used _for_ start up	key_ slot_ used _for_ sync	offset_correction_start													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 318. Protocol Configuration Register 11 (PCR11)

Protocol Configuration Register 12 (PCR12)

Base + 0x00B8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	allow_passive_to_active				key_slot_header_crc											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 319. Protocol Configuration Register 12 (PCR12)

Protocol Configuration Register 13 (PCR13)

Base + 0x00BA — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	first_minislot_action_point_offset								static_slot_after_action_point							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 320. Protocol Configuration Register 13 (PCR13)

Protocol Configuration Register 14 (PCR14)

Base + 0x00BC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	rate_correction_out												listen_timeout[20:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 321. Protocol Configuration Register 14 (PCR14)

Protocol Configuration Register 15 (PCR15)

Base + 0x00BE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 322. Protocol Configuration Register 15 (PCR15)

Protocol Configuration Register 16 (PCR16)

Base + 0x00C0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	macro_initial_offset_b								noise_listen_timeout[24:16]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 323. Protocol Configuration Register 16 (PCR16)

Protocol Configuration Register 17 (PCR17)

Base + 0x00C2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	noise_listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 324. Protocol Configuration Register 17 (PCR17)

Protocol Configuration Register 18 (PCR18)

Base + 0x00C4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	wakeup_pattern								key_slot_id							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 325. Protocol Configuration Register 18 (PCR18)

Protocol Configuration Register 19 (PCR19)

Base + 0x00C6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	decoding_correction_a								payload_length_static							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 326. Protocol Configuration Register 19 (PCR19)

Protocol Configuration Register 20 (PCR20)

Base + 0x00C8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	micro_initial_offset_b								micro_initial_offset_a							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 327. Protocol Configuration Register 20 (PCR20)

Protocol Configuration Register 21 (PCR21)

Base + 0x00CA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	extern_rate_correction				latest_tx											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 328. Protocol Configuration Register 21 (PCR21)

Protocol Configuration Register 22 (PCR22)

Base + 0x00CC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	R*	comp_accepted_startup_range_a											micro_per_cycle[19:16			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 329. Protocol Configuration Register 22 (PCR22)

Protocol Configuration Register 23 (PCR23)

Base + 0x00CE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 330. Protocol Configuration Register 23 (PCR23)

Protocol Configuration Register 24 (PCR24)

Base + 0x00D0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cluster_drift_damping				max_payload_length_dynamic								micro_per_cycle_min [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 331. Protocol Configuration Register 24 (PCR24)

Protocol Configuration Register 25 (PCR25)

Base + 0x00D2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_min[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 332. Protocol Configuration Register 25 (PCR25)

Protocol Configuration Register 26 (PCR26)

Base + 0x00D4 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_halt_due_to_clock				comp_accepted_startup_range_b								micro_per_cycle_max [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 333. Protocol Configuration Register 26 (PCR26)

Protocol Configuration Register 27 (PCR27)

Base + 0x00D6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_max[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 334. Protocol Configuration Register 27 (PCR27)

Protocol Configuration Register 28 (PCR28)

Base + 0x00D8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	dynamic_slot_idle_phase				macro_after_offset_correction											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 335. Protocol Configuration Register 28 (PCR28)

Protocol Configuration Register 29 (PCR29)

Base + 0x00DA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	extern_offset_correction				minislots_max											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 336. Protocol Configuration Register 29 (PCR29)

Protocol Configuration Register 30 (PCR30)

Base + 0x00DC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 337. Protocol Configuration Register 30 (PCR30)

Message Buffer Configuration, Control, Status Registers (MBCCSR*n*)

Base + 0x0100 (MBCCSR0)
Base + 0x0108 (MBCCSR1)
...
Base + 0x02F8 (MBCCSR63)

Write: MCM, MBT, MTD: *POC:config* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MCM	MBT	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 338. Message Buffer Configuration, Control, Status Registers (MBCCSR*n*)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in *Section 21.6.6: Individual message buffer functional description.*

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.

If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 322. MBCCSR*n* field descriptions

Field	Description
Message buffer configuration	
MCM	**Message Buffer Commit Mode** — This bit configures the commit mode of a double buffered message buffer. 0Streaming commit mode 1Immediate commit mode
MBT	**Message Buffer Type** — This bit configures the buffering type of a transmit message buffer. 0 Single buffered message buffer 1 Double buffered message buffer
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer 1 Transmit message buffer

Table 322. MBCCSR*n* field descriptions (continued)

Field	Description
Message buffer control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission. 1 Message buffer data ready for transmission.
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value EDS status bit is 0. 0 No effect. 1 Message buffer enable or disable is triggered.
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect. 1 Message buffer lock or unlock is triggered.
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
Message buffer status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame Header and Message buffer data field not updated 1 Frame Header and Message buffer data field updated
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 receive message buffer contains no valid frame data / message is transmitted for the first time 1 receive message buffer contains valid frame data / message will be transferred again
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event 1 Slot status field updated or transmit message buffer just enabled.

Message Buffer Cycle Counter Filter Registers (MBCCFR*n*)

Base + 0x0102 (MBCCFR0)
Base + 0x010A (MBCCFR1)
...
Base + 0x021FA (MBCCFR63)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MTM	CHA	CHB	CCFE	CCFMSK						CCFVAL					
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 339. Message Buffer Cycle Counter Filter Registers (MBCCFR*n*)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to *Section : Message buffer cycle counter filtering.*

Table 323. MBCCFR*n* field descriptions

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode 1 State transmission mode
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to ***Table 324***.
CCFE	**Cycle Counter Filtering Enable** — This control bit enables and disables the cycle counter filtering. 0 Cycle counter filtering disabled. 1 Cycle counter filtering enabled.
CCFMSK	Cycle Counter Filtering Mask — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 324. Channel assignment description

CHA	CHB	Transmit Message Buffer		Receive Message Buffer	
		static segment	dynamic segment	static segment	dynamic segment
1	1	transmit on both channel A and channel B	transmit on channel A only	store first valid frame received on either channel A or channel B	store first valid frame received on channel A, ignore channel B
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

Note: If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise,

the message buffer configuration is illegal and the result of the message buffer search is not defined.

Message Buffer Frame ID Registers (MBFIDR*n*)

Base + 0x0104 (MBFIDR0)
Base + 0x010C (MBFIDR1)
...
Base + 0x02FC (MBFIDR63)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FID										
W																
Reset	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-

Figure 340. Message Buffer Frame ID Registers (MBFIDR*n*)

Table 325. MBFIDRn field descriptions

Field	Description
FID	**Frame ID** — The semantic of this field depends on the message buffer transfer type. – *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. – *Transmit Message Buffer:* This field determines the slot in which the message in this message buffer should be transmitted.

Message Buffer Index Registers (MBIDXR*n*)

Base + 0x0106 (MBIDXR0)
Base + 0x010E (MBIDXR1)
...
Base + 0x02FE (MBIDXR63)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	MBIDX						
W																
Reset	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-

Figure 341. Message Buffer Index Registers (MBIDXR*n*)

Table 326. MBIDXR*n* field descriptions

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The controller updates this register after frame reception or transmission.

21.6 Functional description

This section provides a detailed description of the functionality implemented in the controller.

21.6.1 Message buffer concept

The controller uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in *Section 21.6.3: Message buffer types.* The physical message buffer is located in the FlexRay memory and is described in *Section 21.6.2: Physical message buffer.*

21.6.2 Physical message buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the FlexRay memory.The structure of a physical message buffer is shown in *Figure 342*.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header*, the *data field offset*, and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

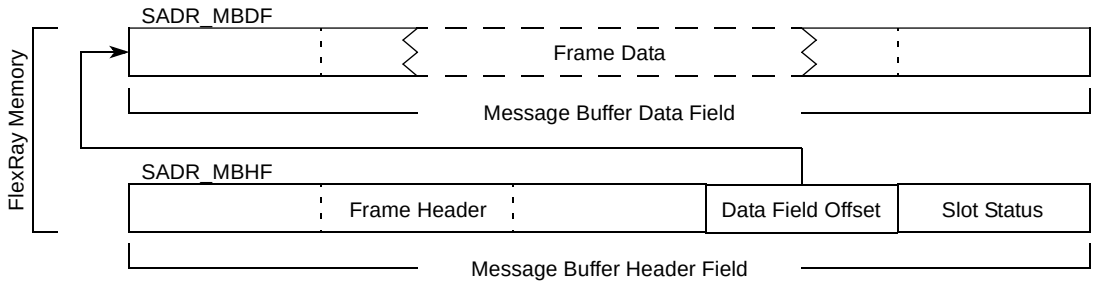


Figure 342. Physical message buffer structure

Message buffer header field

The message buffer header field is a contiguous region in the FlexRay memory and occupies ten bytes. It contains the frame header, the data field offset, and the slot status. Its structure is shown in *Figure 342*. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

Frame header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the usage and

the content of the frame header is provided in *Section : Frame header description.*

Data field offset

The data field offset follows the frame header in the message buffer data field and occupies two bytes. It contains the offset of the corresponding message buffer data field with respect to the controller FlexRay memory base address as provided by SYS_MEM_BASE_ADDR field in the *System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)*. The data field offset determines the start address *SADR_MBDF* of the corresponding message buffer data field in the FlexRay memory according to *Equation 22*.

Equation 22SADR_MBDF = [Data Field Offset] + SYS_MEM_BASE_ADDR

Slot status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in *Section : Slot status description.*

Message buffer data field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in *Section 21.6.3: Message buffer types.*

21.6.3 Message buffer types

The controller provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

Individual message buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The controller supports three types of individual message buffers, which are described in *Section 21.6.6: Individual message buffer functional description.*

Each individual message buffer consists of two parts, the physical message buffer, which is located in the FlexRay memory, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in *Figure 343*.

Each individual message buffer has a message buffer number n assigned, which determines the set of message buffer control registers associated to this individual message

buffer. The individual message buffer with message buffer number *n* is controlled by the registers MBCCSRn, MBCCFRn, MBFIDRn, and MBIDXRn.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the *Message Buffer Index Registers (MBIDXRn)*. The start address SADR_MBHF of the related message buffer header field in the FlexRay memory is determined according to *Equation 23*.

Equation 23SADR_MBHF = (MBIDXRn[MBIDX] × 10) + SYS_MEM_BASE_ADDR

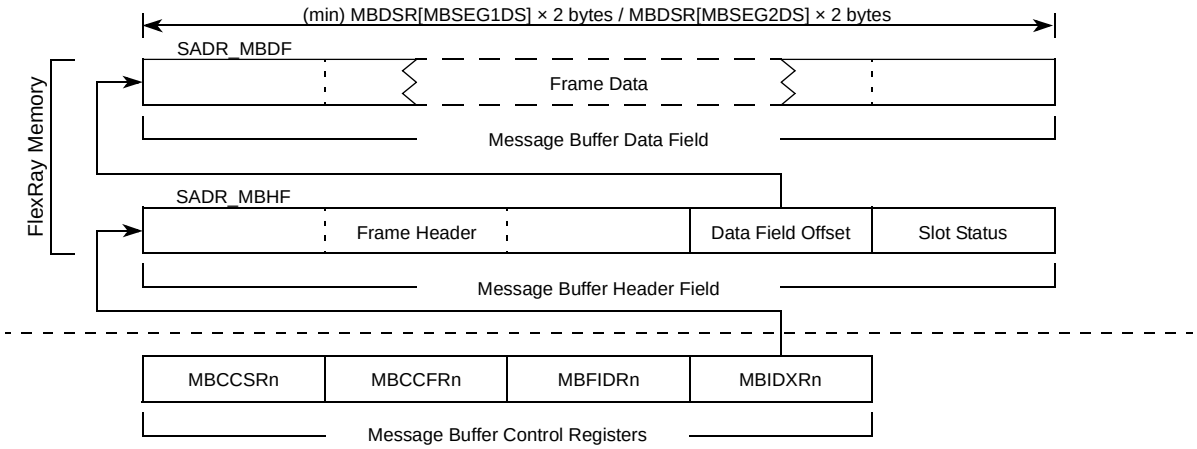


Figure 343. Individual message buffer structure

Individual message buffer segments

The set of the individual message buffers can be split up into two message buffer segments using the *Message Buffer Segment Size and Utilization Register (MBSSUTR)*. All individual message buffers with a message buffer number $n \leq$ MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number $n >$ MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- All physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length.
- The minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 × MBDSR[MBSEG1DS] bytes.
- The minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 × MBDSR[MBSEG2DS] bytes.

Receive shadow buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The controller provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the FlexRay memory and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in *Figure 344*. The four internal

shadow buffer control registers can be accessed by the *Receive Shadow Buffer Index Register (RSBIR)*.

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the *Receive Shadow Buffer Index Register (RSBIR)*. The start address SADR_MBHF of the related message buffer header field in the FlexRay memory is determined according to *Equation 24*.

Equation 24SADR_MBHF = (RSBIR[RSBIDX] × 10) + SYS_MEM_BASE_ADDR

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in *Receive Shadow Buffer Index Register (RSBIR)*.

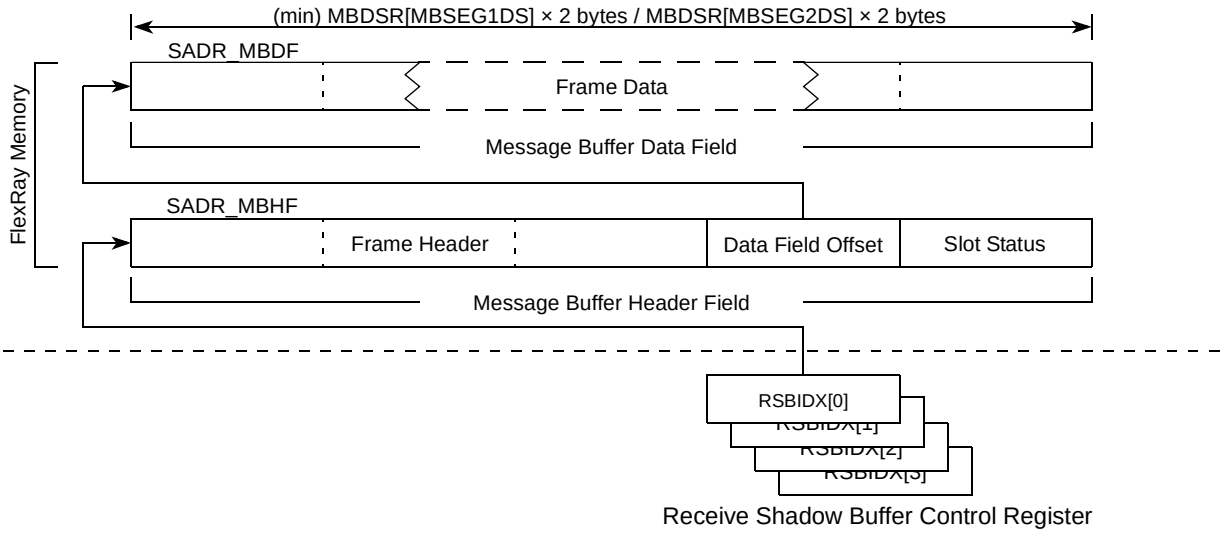


Figure 344. Receive shadow buffer structure

Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The controller provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the FlexRay memory and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in *Figure 345*.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the start index field SIDX in the *Receive FIFO Start Index Register (RFSIR)*, the FIFO depth field FIFO_DEPTH in the *Receive FIFO Depth and Size Register (RFDSR)*, and the read index field RDIDX *Receive FIFO A Read Index Register (RFARIR)* / *Receive FIFO B Read Index Register (RFBRIR)*. The start address SADR_MBHF_1 of the

first message buffer header field that belongs to the receive FIFO in the FlexRay memory is determined according to *Equation 25*.

Equation 25SADR_MBHF[1] = (RFSIR[SIDX] × 10) + SYS_MEM_BASE_ADDR

The start address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the FlexRay memory is determined according to *Equation 26*.

Equation 26SADR_MBHF[n] = ((RFSIR[SIDX] + RFDSR[FIFO_DEPTH]) × 10) + SYS_MEM_BASE_ADDR

Note: All message buffer header fields assigned to a receive FIFO must be a contiguous region.

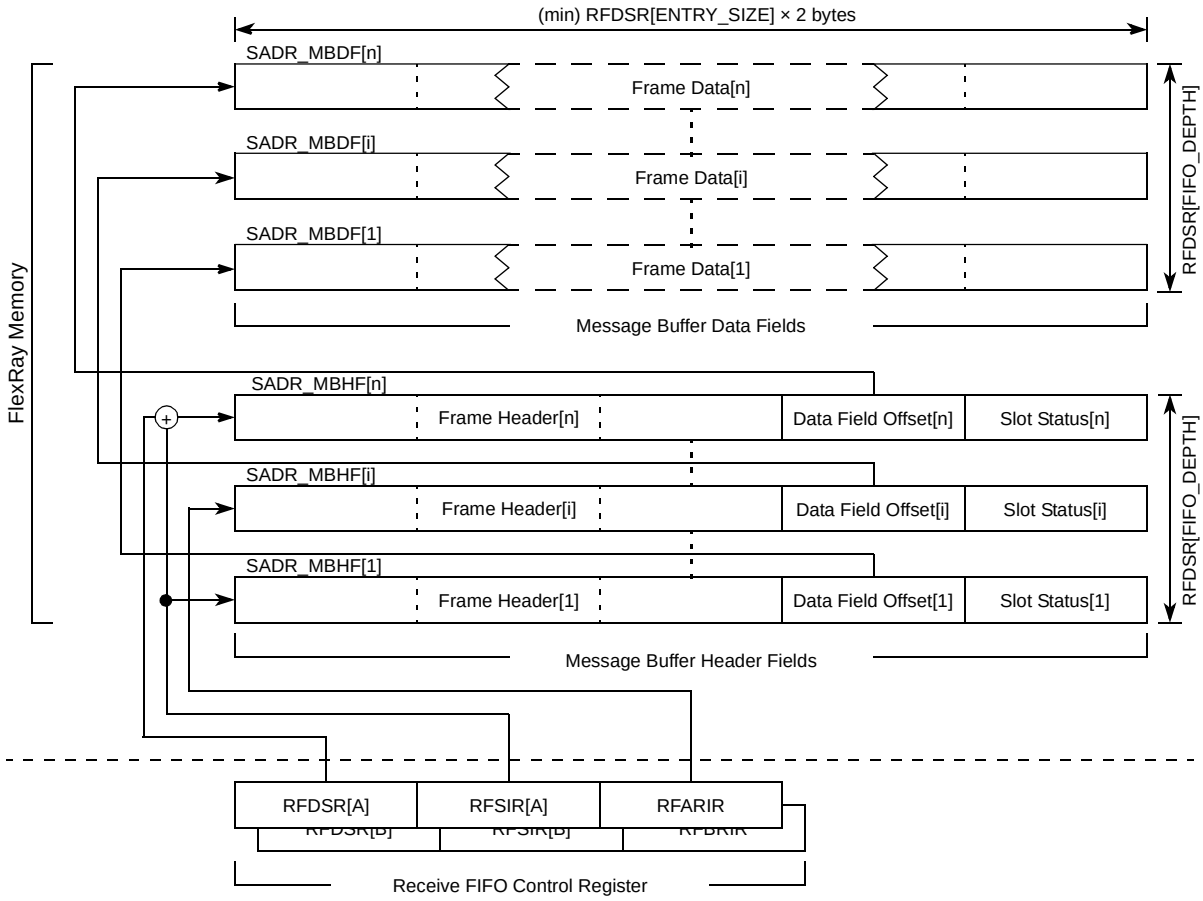


Figure 345. Receive FIFO structure

Message buffer configuration and control data

This section describes the configuration and control data for each message buffer type.

Individual message buffer configuration data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual

message buffers and a set of configuration parameters that applies to each message buffer individually.

0.0.0.0.0.1 Common configuration data

The set of common configuration data for individual message buffers is located in the following registers.

- *Message Buffer Data Size Register (MBDSR)*
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- *Message Buffer Segment Size and Utilization Register (MBSSUTR)*
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see *Section : Individual message buffer segments.*

0.0.0.0.0.2 Specific configuration data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*
  The MCM, MBT, MTD bits configure the message buffer type.
- *Message Buffer Cycle Counter Filter Registers (MBCCFRn)*
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- *Message Buffer Frame ID Registers (MBFIDRn)*
  For a transmit message buffer, the FID field determines the slot in which the message in this message buffer will be transmitted.
- *Message Buffer Index Registers (MBIDXRn)*
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

Individual message buffer control data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*.

Receive shadow buffer configuration data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the *Receive Shadow Buffer Index Register (RSBIR)*. For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full controller control.

Receive FIFO control and configuration data

This section describes the configuration and control data for the two receive FIFOs.

Receive FIFO configuration data

The controller provides two completely independent receive FIFOs, one per channel. Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- *Receive FIFO Start Index Register (RFSIR)*
- *Receive FIFO Depth and Size Register (RFDSR)*
- *Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)*
- *Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)*
- *Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)*
- *Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)*
- *Receive FIFO Range Filter Configuration Register (RFRFCFR)*

Receive FIFO control data

The application can access the receive FIFO at any time using the values provided in the *Receive FIFO A Read Index Register (RFARIR)* and *Receive FIFO B Read Index Register (RFBRIR)*. To update the *Receive FIFO A Read Index Register (RFARIR)*, the application must write 1 to the FIFO A Not Empty Interrupt Flag FNEAIF in the *Global Interrupt Flag and Enable Register (GIFER)*. To update the *Receive FIFO B Read Index Register (RFBRIR)* the application must write 1 to the FIFO B Not Empty Interrupt Flag FNEBIF in the *Global Interrupt Flag and Enable Register (GIFER)*. As long as the FIFO is not empty, each update increments the related read index. If the read index has reached the last FIFO entry, it wraps back to the FIFO start index.

21.6.4 FlexRay memory layout

The controller supports a wide range of possible layouts for the FlexRay memory. *Figure 346* shows an example layout. The following set of rules applies to the layout of the FlexRay memory:

- The FlexRay memory is a contiguous region.
- The FlexRay memory size is maximum 64 KB.
- The FlexRay memory starts at a 16 byte boundary.

The FlexRay memory contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*. The areas are described in this section.

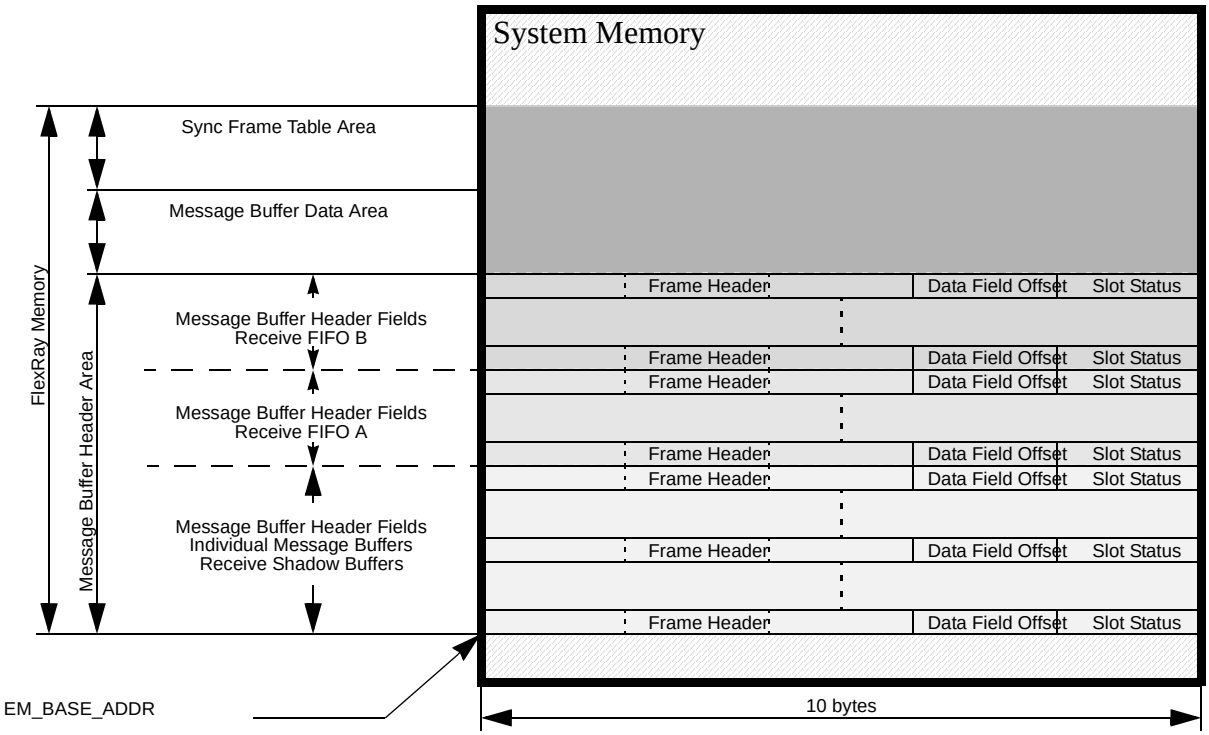


Figure 346. Example of FlexRay memory layout

Message buffer header area

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill *Equation 27*.

Equation 27SADR_MBHF = (i × 10) + SYS_MEM_BASE_ADDR; (0 <= i < 64)

– The start address SADR_MBHF of each message buffer header field for the *receive FIFO* must fulfill *Equation 28*.

Equation 28SADR_MBHF = (i × 10) + SYS_MEM_BASE_ADDR; (0 <= i < 1024)

– The message buffer header fields for a receive FIFO have to be a contiguous area.

Message buffer data area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

Sync frame table area

The sync frame table area provides a copy of the internal sync frame tables for application access. Refer to *Section 21.6.12: Sync frame ID and sync frame deviation tables* for the description of the sync frame table area.

21.6.5 Physical message buffer description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

Message buffer protection and data consistency

The physical message buffers are located in the FlexRay memory. The controller provides no means to protect the FlexRay memory from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

Message buffer header field description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in *Section : Message buffer header field.* Each message buffer header field consists of three sections: the frame header section, the data field offset, and the slot status section. For a detailed description of the Data Field Offset, see *Section : Data field offset.*

Frame header description

0.0.0.0.0.3 Frame header content

The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in *Figure 347*. A detailed description is given in *Table 328*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	R	PPI	NUF	SYF	SUF	FID										
0x2	0	0	CYCCNT						0	PLDLEN						
0x4	0	0	0	0	0	HDCRC										

Figure 347. Frame header structure (Receive message buffer and receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in *Figure 348*. A detailed description is given in *Table 329*. The checks that will be performed are described in *Frame header checks*.

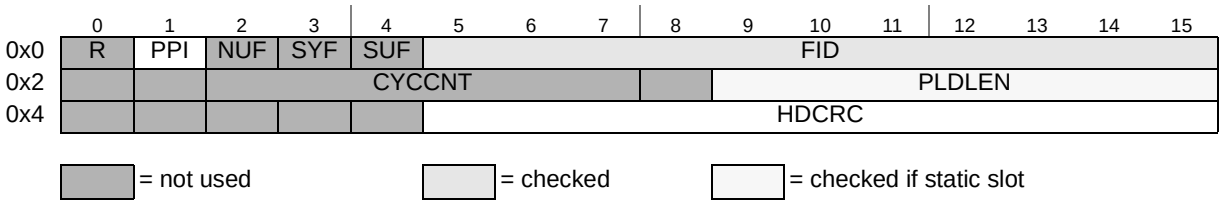


Figure 348. Frame header structure (Transmit message buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in *Figure 349*.

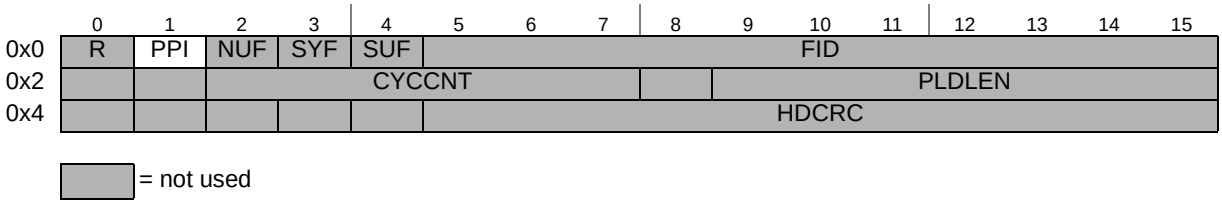


Figure 349. Frame header structure (Transmit message buffer for key slot)

0.0.0.0.0.4 Frame header access

The frame header is located in the FlexRay memory. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 327*. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 327. Frame header write access constraints (Transmit message buffer)

Field	Single Buffered		Double Buffered			
	Static Segment	Dynamic Segment	Static Segment		Dynamic Segment	
			Commit Side	Transmit Side	Commit Side	Transmit Side
FID	*POC:config* or MB_DIS					
PPI, PLDLEN, HDCRC	*POC:config* or MB_DIS or	MB_LCK			MB_LCK	

0.0.0.0.0.5 Frame header checks

As shown in *Figure 348* and *Figure 349* not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored,

some are used for transmission, and some of them are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding *Message Buffer Frame ID Registers (MBFIDRn)*. If the controller detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the *CHI Error Flag Register (CHIERFR)*. The value of the FID field will be ignored and replaced by the value provided in the *Message Buffer Frame ID Registers (MBFIDRn)*.

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the *Protocol Configuration Register 19 (PCR19)*. If this is not fulfilled, the static payload length error flag SPL_EF in the *CHI Error Flag Register (CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the *Protocol Configuration Register 24 (PCR24)*. If this is not fulfilled, the dynamic payload length error flag DPL_EF in the *CHI Error Flag Register (CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 328. Frame header field descriptions (Receive message buffer and receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer.
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 329. Frame header field descriptions (Transmit message buffer)

Field	Description
R	**Reserved Bit** — This bit is not used, the value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used, the value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
SYF	**Sync Frame Indicator** — This bit is not used, the value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
SUF	**Startup Frame Indicator** — This bit is not used, the value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
FID	**Frame ID** — This field is checked as described in *Frame header checks*.
CYCCNT	**Cycle Count** — This field is not used, the value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.
PLDLEN	**Payload Length** — This field is checked and used as described in *Frame header checks*.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

Data field offset description

0.0.0.0.0.6 Data field offset content

For a detailed description of the Data Field Offset, see *Section : Data field offset.*

0.0.0.0.0.7 Data field offset access

The application shall program the Data Field Offset when configuring the message buffers either in the *POC:config* state or when the message buffer is disabled.

Slot status description

The slot status is a read-only structure for the application and a write-only structure for the controller. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

0.0.0.0.0.8 Receive message buffer and receive FIFO slot status description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by *Table 330*.

Table 330. Receive message buffer slot status content

Receive Message Buffer Type	Slot Status Content
Individual Receive Message Buffer assigned to both channels MBCCSRn[CHA]=1 and MBCCSRn[CHB]=1	see *Figure 350*
Individual Receive Message Buffer assigned to channel A MBCCSRn[CHA]=1 and MBCCSRn[CHB[=0	see *Figure 351*
Individual Receive Message Buffer assigned to channel B MBCCSRn[CHA]=0 and MBCCSRn[CHB]=1	see *Figure 352*
Receive FIFO Channel A Message Buffer	see *Figure 351*
Receive FIFO Channel B Message Buffer	see *Figure 352*

The meaning of the bits in the slot status structure is explained in *Table 331*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 350. Receive message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 351. Receive message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 352. Receive message buffer slot status structure (ChB)

Table 331. Receive message buffer slot status field descriptions

Field	Description
	Common Message Buffer Status Bits
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.

Table 331. Receive message buffer slot status field descriptions (continued)

Field	Description
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
CH	**Channel first valid received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all. 0 first valid frame received on channel B.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!Vali. dFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.

0.0.0.0.0.9 Transmit message buffer slot status description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the

slot status structure for transmit message buffers depends on the channel assignment as given by *Table 332*.

Table 332. Transmit message buffer slot status content

Transmit Message Buffer Type	Slot Status Content
Individual Transmit Message Buffer assigned to both channels MBCCSRn[CHA]=1 and MBCCSRn[CHB]=1	see *Figure 353*
Individual Transmit Message Buffer assigned to channel A MBCCSRn[CHA]=1 and MBCCSRn[CHB]=0	see *Figure 354*
Individual Transmit Message Buffer assigned to channel B MBCCSRn[CHA]=0 and MBCCSRn[CHB]=1	see *Figure 355*

The meaning of the bits in the slot status structure is described in *Table 331*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 353. Transmit message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 354. Transmit message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 355. Transmit message buffer slot status structure (ChB)

Table 333. Transmit message buffer slot status structure field descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.

Table 333. Transmit message buffer slot status structure field descriptions (continued)

Field	Description
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

Message buffer data field description

The message buffer data field stores the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in *Table 334*. The structure of the message buffer data field is given in *Figure 356*.

Table 334. Message buffer data field minimum length

physical message buffer assigned to	minimum length defined by
Individual Message Buffer in Segment 1	MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	MBDSR[MBSEG2DS]
Receive FIFO for channel A	RFDSR[ENTRY_SIZE] (RFSR[SEL] = 0)
Receive FIFO for channel B	RFDSR[ENTRY_SIZE] (RFSR[SEL] = 1)

Note: The controller will not access any locations outside the message buffer data field boundaries given by Table 334.

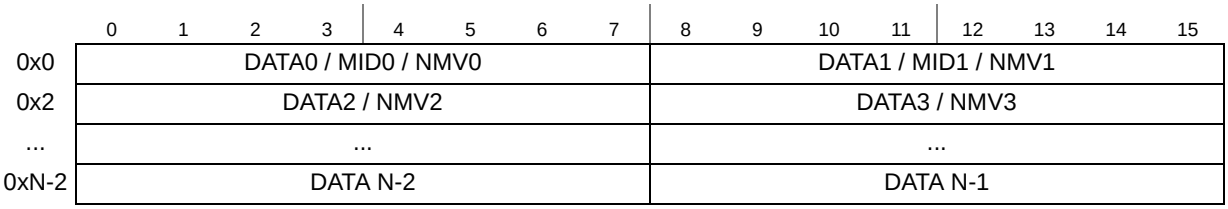


Figure 356. Message buffer data field structure

The message buffer data field is located in the FlexRay memory; thus, the controller has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

Message buffer data field read access

For transmit message buffers, the controller will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the *Message Buffer Index Registers (MBIDXRn)*. While the message buffer is locked, the controller will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the *Receive FIFO A Read Index Register (RFARIR)* or the *Receive FIFO B Read Index Register (RFBRIR)* when the related receive FIFO non-empty interrupt flag FNEAIF or FNEBIF is set in the *Global Interrupt Flag and Enable Register (GIFER)*. While the non-empty interrupt flag is set, the controller will not update the Message Buffer Data Field related to message buffer indicated by *Receive FIFO A Read Index Register (RFARIR)* or the *Receive FIFO B Read Index Register (RFBRIR)*.

Message buffer data field write access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 335*.

Table 335. Frame data write access constraints

Field	Single buffered	Double buffered	
		Commit side	Transmit side
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS

Table 336. Frame data field descriptions

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	Message Identifier — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	Network Management Vector — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. The MID and NMV bytes replace the corresponding DATA bytes.

21.6.6 Individual message buffer functional description

The controller provides three basic types of individual message buffers:

1. Single Transmit Message Buffers
   - Double Transmit Message Buffers
   - Receive Message Buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in *Section : Individual message buffer configuration data.*

Individual message buffer configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the FlexRay memory. The second step is the programming of the message buffer configuration registers, which is described in this section.

Common configuration data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the *Message Buffer Segment Size and Utilization Register (MBSSUTR)*.

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the *Message Buffer Segment Size and Utilization Register (MBSSUTR)*

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the *Message Buffer Data Size Register (MBDSR)*.

Depending on the current receive functionality of the controller, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the *Receive Shadow Buffer Index Register (RSBIR)*.

Specific configuration data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled, i.e., MBCCSR*n*[EDS] = 0

The individual message buffer type is defined by the MTD and MBT bits in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)* as given in *Table 337*.

Table 337. Individual message buffer types

MBCCSRn[MTD]	MBCCSRn[MBT]	Individual Message Buffer Description
0	0	Receive Message Buffer
0	1	Reserved
1	0	Single Transmit Message Buffer
1	1	Double Transmit Message Buffer

The message buffer specific configuration data are

1. MCM, MBT, MTD bits in *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*
   - all fields and bits in *Message Buffer Cycle Counter Filter Registers (MBCCFRn)*
   - all fields and bits in *Message Buffer Frame ID Registers (MBFIDRn)*
   - all fields and bits in *Message Buffer Index Registers (MBIDXRn)*

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions *Section : Single transmit message buffers, Section : Receive message buffers* and *Section : Double transmit message buffer.*

Single transmit message buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A single transmit message buffer is used by the application to provide message data to the controller that will be transmitted over the FlexRay Bus. The controller uses the transmit message buffers to provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a single transmit message buffer by the following settings:

- MBCCSRn[MBT] = 0 (single buffered message buffer)
- MBCCSRn[MTD] = 1 (transmit message buffer)

Access regions

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the access to the data, control, and status bits of a message buffer. The access regions for single transmit message buffers are shown in *Figure 357*. A description of the regions is given in *Table 338*. If an region is active as indicated in *Table 339*, the access scheme given for that region applies to the message buffer.

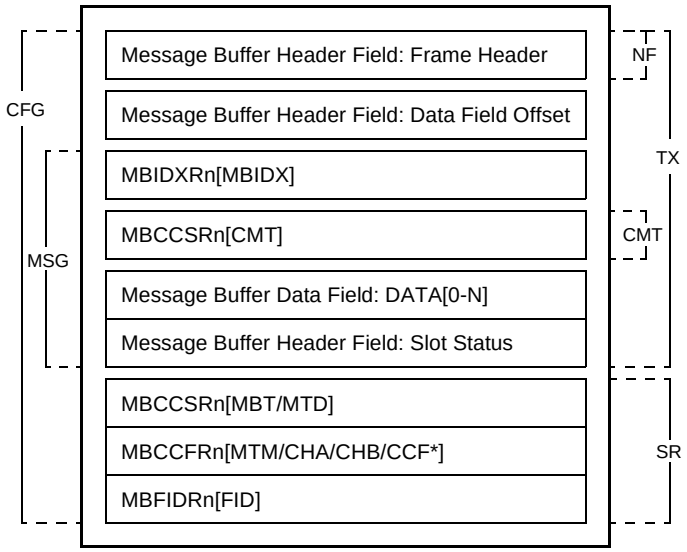


Figure 357. Single transmit message buffer access regions

Table 338. Single transmit message buffer access regions description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Data and Slot Status Access
NF	—	read-only	Message Header Access for Null Frame Transmission
TX	—	read/write	Message Transmission and Slot Status Update
CM	—	read-only	Message Buffer Validation
SR	—	read-only	Message Buffer Search

The trigger bits MBCCSRn[EDT] and MBCCSRn[LCKT], and the interrupt enable bit MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSnR[MBIF] is not under access control and can be accessed from the application and the controller at any time. controller clear access has higher priority.

The controller restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in *Figure 358*. A description of the states is given in *Table 339*, which also provides the access scheme for the access regions.

The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

Message buffer states

This section describes the transmit message buffer states and provides a state diagram.

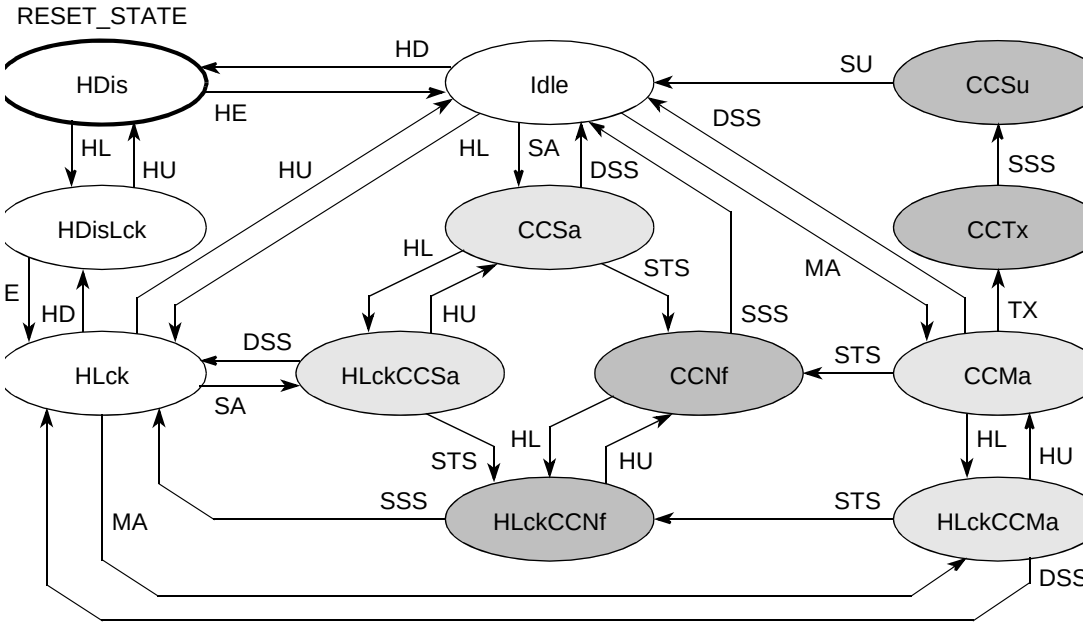


Figure 358. Single transmit message buffer states

Table 339. Single transmit message buffer state description

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	CM, SR	**Idle** — Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** — Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	Locked — Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	—	—	Slot Assigned — Message buffer assigned to next static slot. Ready for Null Frame transmission.

Table 339. Single transmit message buffer state description (continued)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
HLckCCSa	1	1	MSG	—	Locked and Slot Assigned — Applications access to data, control, and status. Message buffer assigned to next static slot
CCNf	1	0	—	NF	Null Frame Transmission Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	Locked and Null Frame Transmission — Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	—	CM	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	—	Locked and Message Available — Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	—	TX	Message Transmission — Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	—	TX	Status Update — Message buffer status update. Update of status flags, the slot status field, and the header index.

Message buffer transitions

0.0.0.0.0.10 Application transitions

The application transitions can be triggered by the application using the commands described in *Table 340*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSRn[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (CHIERFR)* is set.

Table 340. Single transmit message buffer application transitions

Transition	Command	Condition	Description
HE	MBCCSRn[EDT]:= 1	MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSRn[LCKT]:= 1	MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

0.0.0.0.0.11 Module transitions

The module transitions that can be triggered by the controller are described in *Table 341*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 341. Single transmit message buffer module transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned — Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSRn[CMT] = 1	Transmission Slot Start — Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated — Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start — Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start — Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start — Start of static slot or dynamic slot or symbol window or NIT.

0.0.0.0.0.12 Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 342*, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of *Table 342*.

Table 342. Single transmit message buffer transition priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

Transmit message setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn).* The physical access to the message buffer data field is described in *Section : Individual message buffers.*

As indicated by *Table 339*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 340*. The state change is indicated through the MBCCSRn[EDS] and MBCCSRn[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the MBCCSRn[DVAL] flag is negated.

Message transmission

As a result of the message buffer search described in *Section 21.6.7: Individual message buffer search* the controller triggers the message available transition MA for as many as two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The controller transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
   – the message data are still valid, i.e., MBCCSR*n*[CMT] = 1

In this case, the controller triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in *Figure 359*. In this example, the message buffer with message buffer number n is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid, i.e., MBCCSR*n*[CMT] = 1.

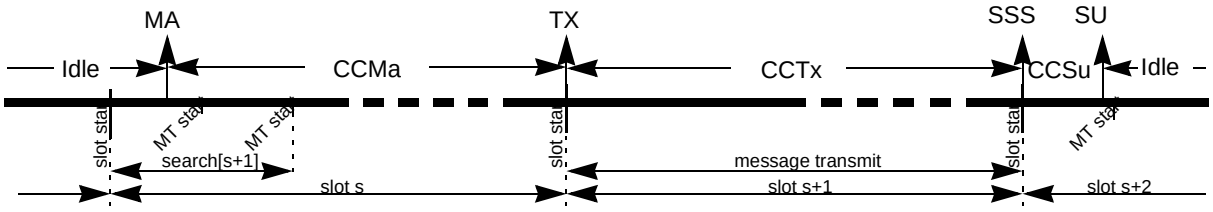


Figure 359. Message transmission timing

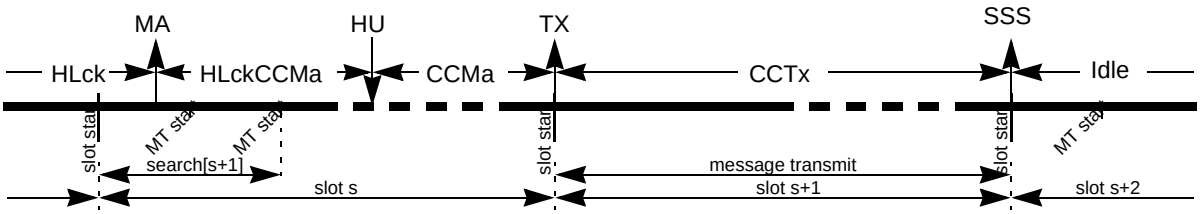


Figure 360. Message transmission from HLck state with unlock

The amount of message data read from the FlexRay memory and transferred to the FlexRay bus is determined by the following three items:

1. the message buffer segment that the message buffer is assigned to, as defined by MBSSUTR (see *Section : Message Buffer Segment Size and Utilization Register (MBSSUTR)"*)
   - the message buffer data field size, as defined by the related field of MBDSR (see *Section : Message Buffer Data Size Register (MBDSR)"*)
   - the value of the PLDLEN field in the message buffer header field, as described in *Section : Frame header description.*

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 × MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN <= MBSEG1DS, the controller reads and transfers 2 × PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

Null frame transmission

A static slot with slot number S is assigned to the controller for channel A, if at least one transmit message buffer is configured with the MBFIDR*n*[FID] set to S and MBCCFRn[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the controller for channel A, and all transmit message buffers with MBFIDR*n*[FID] = s and MBCCFR*n*[CHA] = 1 are either not committed, i.e., MBCCSR*n*[CMT] = 0, or locked by the application, i.e., MBCCSR*n*[LCKS] = 1, or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in *Section 21.6.7: Individual message buffer search* the controller triggers the slot assigned transition SA for as many as two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in *Figure 361*.

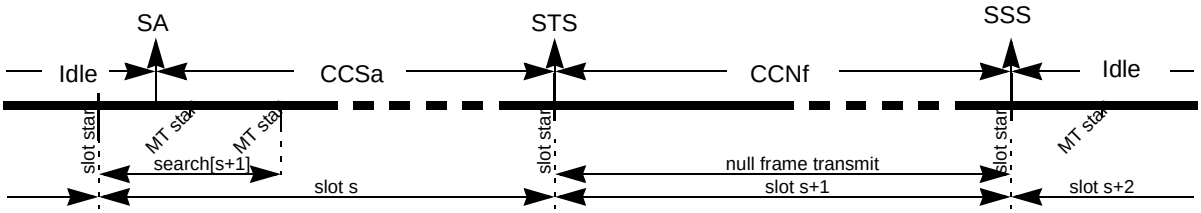


Figure 361. Null frame transmission from idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in *Figure 362*.

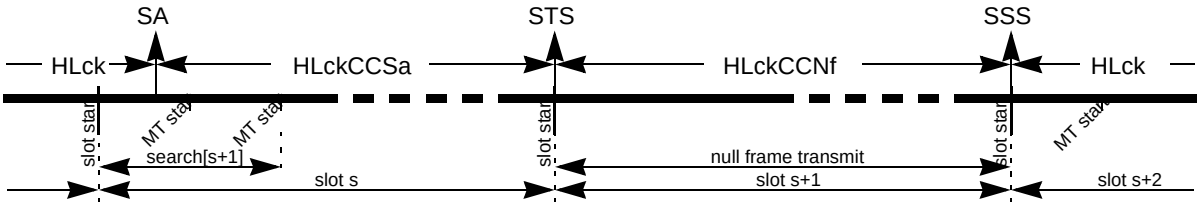


Figure 362. Null frame transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 363*.

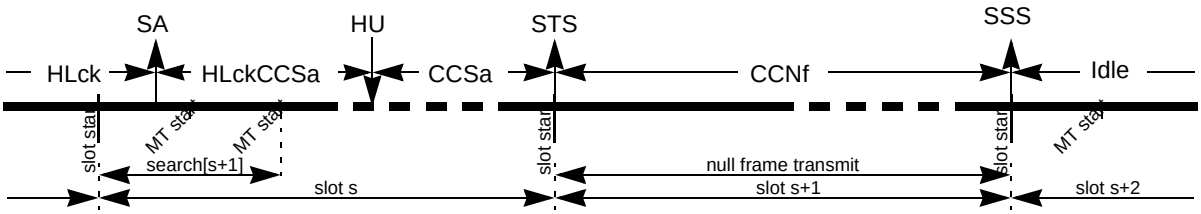


Figure 363. Null frame transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 364*.

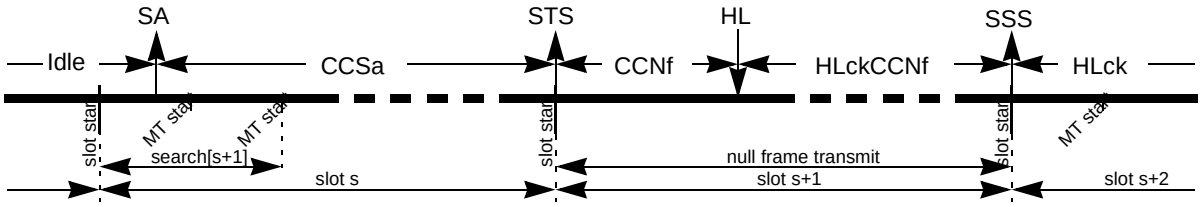


Figure 364. Null frame transmission from idle state with locking

Message buffer status update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

0.0.0.0.0.13 Message buffer status update after complete message transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the controller updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the controller sets the message buffer interrupt flag MBCCSn[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag MBCCFRn[MTM], the controller changes the commit flag MBCCSRn[CMT] and the valid flag MBCCSRn[DVAL]. If the MBCCFRn[MTM] flag is negated, the message buffer is in the *event transmission mode.* In this case, each committed message is transmitted only once. The commit flag MBCCSRn[CMT] is cleared with the SU transition. If the MBCCFRn[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The controller will not clear the MBCCSRn[CMT] flag at the end of transmission and will set the valid flag MBCCSRn[DVAL] to indicate that the message will be transmitted again.

0.0.0.0.0.14 Message buffer status update after incomplete message transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
   – The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the *Protocol Interrupt Flag Register 1 (PIFR1)*.

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

0.0.0.0.0.15 Message buffer status update after null frame transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Receive message buffers

The section provides a detailed description of the functionality of the receive message buffers.

A receive message buffer receives a message from the FlexRay Bus based on individual filter criteria. The controller uses the receive message buffer to provide the following data to the application

1. message data received
   - information about the reception process
   - status information about the slot in which the message was received

A individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings

- MBCCSRn[MBT] = 0 (single buffered message buffer)
- MBCCSRn[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented that controls the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are shown in *Figure 365*. A description of the regions is given in *Table 343*. If an region is active as indicated in *Table 344*, the access scheme given for that region applies to the message buffer.

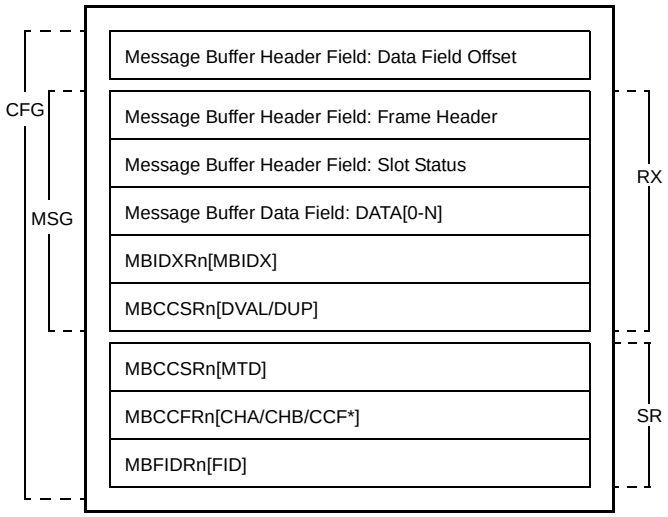


Figure 365. Receive message buffer access regions

Table 343. Receive message buffer access region description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	—	Message Data, Header, and Status Access
RX	—	write-only	Message Reception and Status Update
SR	—	read-only	Message Buffer Search Data

The trigger bits MBCCSRn[EDT] and MBCCSRn[LCKT] and the interrupt enable bit MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSRn[MBIF] is not under access control and can be accessed from the application and the controller at any time. controller set access has higher priority.

The controller restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in *Figure 366*. A description of the message buffer states is given in *Table 339*, which also provides the access scheme for the access regions.

The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

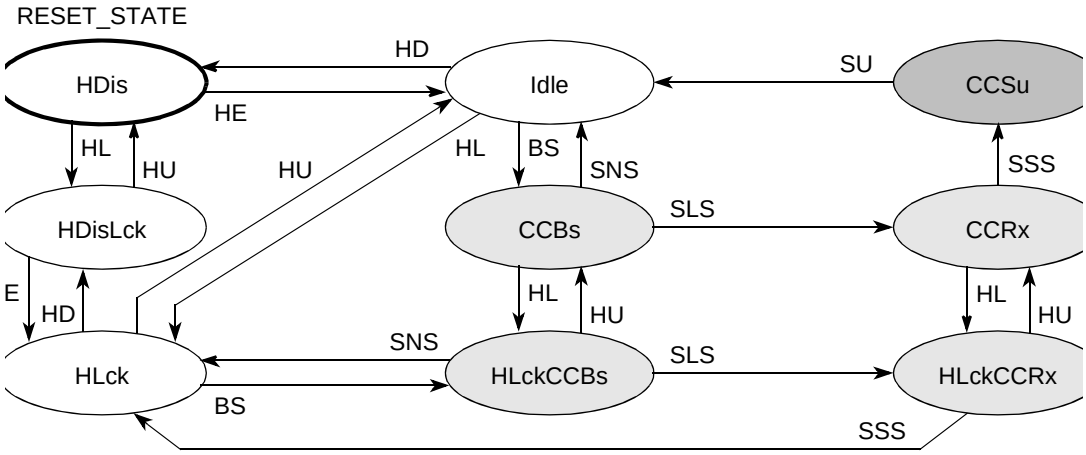


Figure 366. Receive message buffer states

Table 344. Receive message buffer states and access

State	MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	SR	**Idle** — Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** — Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	—	**Locked** — Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	—	—	**Buffer Subscribed** — Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.

Table 344. Receive message buffer states and access (continued)

State	MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
HLckCCBs	1	1	MSG	—	**Locked and Buffer Subscribed** — Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	—	—	**Message Receive** — Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	—	Locked and Message Receive — Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	—	RX	**Status Update** — Message buffer status update. Update of status flags, the slot status field, and the header index.

Message buffer transitions

0.0.0.0.0.16 Application transitions

The application transitions that can be triggered by the application using the commands described in *Table 345*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSRn[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (CHIERFR)* is set.

Table 345. Receive message buffer application transitions

Transition	Host Command	Condition	Description
HE	MBCCSRn[EDT]:= 1	MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSRn[LCKT]:= 1	MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

0.0.0.0.0.17 Module transitions

The module transitions that can be triggered by the controller are described in *Table 346*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 346. Receive message buffer module transitions

Transition	Condition	Description
BS	slot match and CycleCounter match	Buffer Subscribed — The message buffer filter matches next slot and cycle.
SLS	slot start	Slot Start — Start of either Static Slot or Dynamic Slot.
SNS	symbol window start or NIT start	Symbol Window or NIT Start — Start of either Symbol Window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start — Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	status updated	Status Updated — Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

0.0.0.0.0.18 Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in *Table 347*, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 347. Receive message buffer transition priorities

State	Priorities	Description
module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

Message reception

As a result of the message buffer search, the controller changes the state of as many as two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see *Section : Receive Shadow Buffers Concept.* The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

Message buffer update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the *CHI Error Flag Register (CHIERFR)*.

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in *Table 348*.

Table 348. Receive message buffer update

vSS!ValidFrame	vRF!Header!NFIndicator	Update description
1	1	Valid non-null frame received. - Message Buffer Data Field updated. - Frame Header Field updated. - Slot Status Field updated. - DUP:= 1 - DVAL:= 1 - MBIF:= 1
1	0	Valid null frame received. - Message Buffer Data Field *not* updated. - Frame Header Field *not* updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed - MBIF:= 1

Table 348. Receive message buffer update (continued)

vSS!ValidFrame	vRF!Header!NFIndicator	Update description
0	x	No valid frame received. - Message Buffer Data Field not updated. - Frame Header Field not updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed. - MBIF:= 1, if the slot was not an empty dynamic slot. An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

Note: If the number of the last slot in the current communication cycle on a given channel is n, then all receive message buffers assigned to this channel with MBFIDRn[FID] > n will not be updated at all.

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in *Figure 367*.

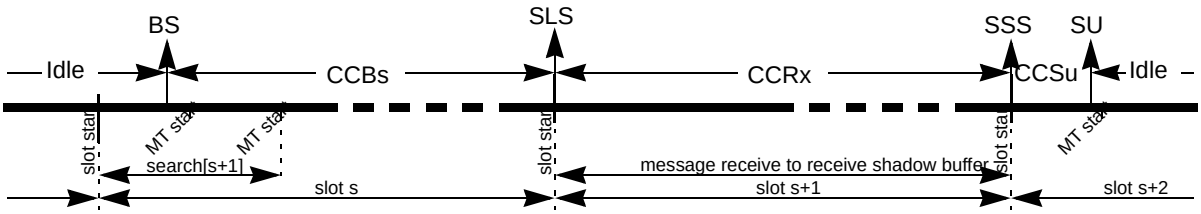


Figure 367. Message reception timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following three items:

1. the message buffer segment that the message buffer is assigned to, as defined by the *Message Buffer Segment Size and Utilization Register (MBSSUTR)*
   - the message buffer data field size, as defined by the related field of the *Message Buffer Data Size Register (MBDSR)*
   - the number of bytes received over the FlexRay bus

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2*MBDSR.MBSEG1DS, the controller writes only 2*MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2*MBDSR.MBSEG1DS, the controller writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with MBDSR.MBSEG2DS.

Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in *Section : Individual message buffers.*

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the controller will not change the content of the message buffer.

Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in *Section : Receive shadow buffers.*

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see *Table 348*), the controller writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the *Message Buffer Index Registers (MBIDXRn)* with the content of the corresponding internal shadow buffer index register. In all other cases, the controller writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the FlexRay memory located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the FlexRay memory accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the *Message Buffer Index Registers (MBIDXRn)*. Instead, the index of the message buffer header field must be fetched from the *Message Buffer Index Registers (MBIDXRn)*.

Double transmit message buffer

The section provides a detailed description of the functionality of the double transmit message buffers.

Double transmit message buffers are used by the application to provide the controller with the message data to be transmitted over the FlexRay Bus. The controller uses this message buffer to provide information to the application about the transmission process, and status information about the slot in which message data was transmitted.

In contrast to the single transmit message buffers, the application can provide new transmission data while the transmission of the previously provided message data is running. This scheme is called double buffering and can be considered as a FIFO of depth 2.

Double transmit message buffers are implemented by combining two individual message buffers that form the two sides of an double transmit message buffer. One side is called the *commit side* and will be accessed by the application to provide the message data. The other side is called the *transmit side* and is used by the controller to transmit the message data to the FlexRay bus. The two sides are located in adjacent individual message buffers. The message buffer that implements the commit side has an even message buffer number 2n.

The transmit side message buffer follows the commit side message buffer and has the message buffer number 2n+1. The basic structure and data flow of a double transmit message buffer is given in *Figure 368*.

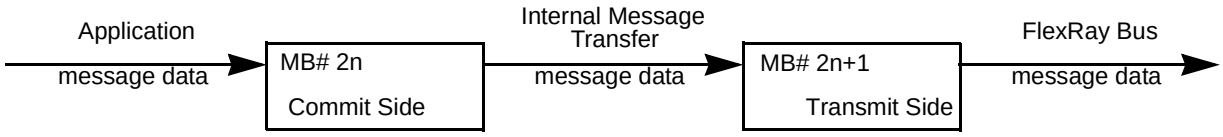


Figure 368. Double transmit buffer structure and data flow

Note: Both the commit and the transmit side must be configured with identical values except for the Message Buffer Index Registers (MBIDXRn).

Access Regions

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the exclusive access to the data, control, and status bits of the message buffer.

The access scheme for double transmit message buffers is shown in *Figure 369*. The given regions represent fields that can be accessed from both the application and the controller and, thus, require access restrictions. A description of the regions is given in *Table 349*.

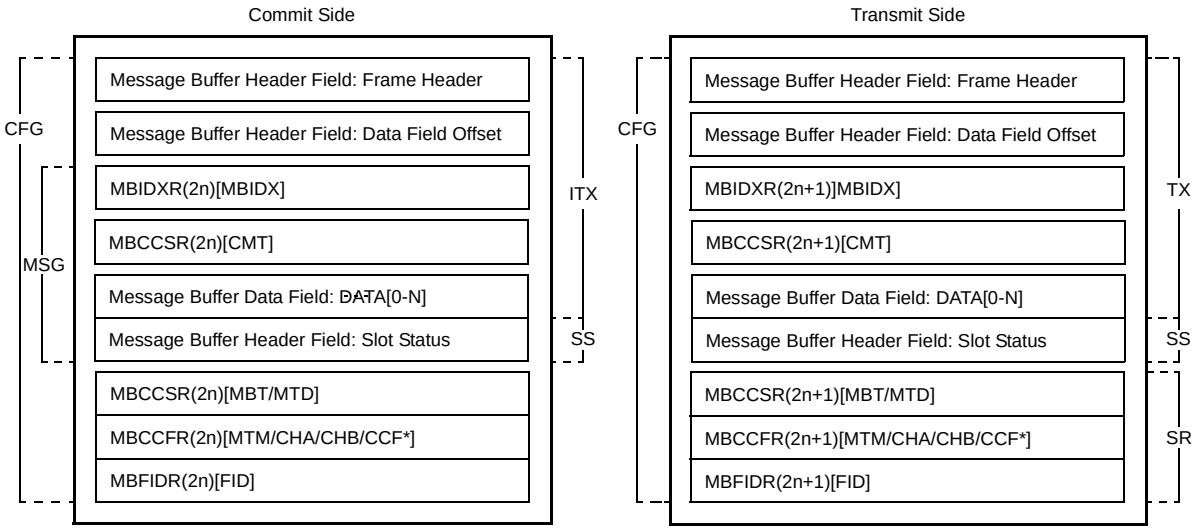


Figure 369. Double transmit message buffer access regions layout

Table 349. Double transmit message buffer access regions description

Access			Description
Region	Type		
	Application	Module	
Commit Side			
CFG	read/write	—	Message Buffer Configuration

Table 349. Double transmit message buffer access regions description (continued)

Access			Description
Region	Type		
	Application	Module	
MSG	read/write	—	Message Buffer Data and Control access
ITX	—	read/write	Internal Message Transfer.
SS	—	write-only	Slot Status Update
Transmit Side			
CFG	read/write	—	Message Buffer Configuration
SR	—	read-only	Message Buffer Search
TX	—	read-only	Internal Message Transfer, Message Transmission
SS	—	write-only	Slot Status Update

The trigger bits MBCCSRn[EDT] and MBCCSRn[LCKT], and the interrupt enable bit MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSnR.MBIF is not under access control and can be accessed from the application and the controller at any time. controller set access has higher priority.

The controller restricts its access to the regions, depending on the current state of the corresponding part of the double transmit message buffer. The application must adhere to these restrictions in order to ensure data consistency. The states for the commit side of a double transmit message buffer are given in *Figure 370*. A description of the states is given in *Table 351*. The states for the transmit side of a double transmit message buffer are given in *Figure 371*. A description of the states is given in *Table 351*. The description tables also provide the access scheme for the access regions.

The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

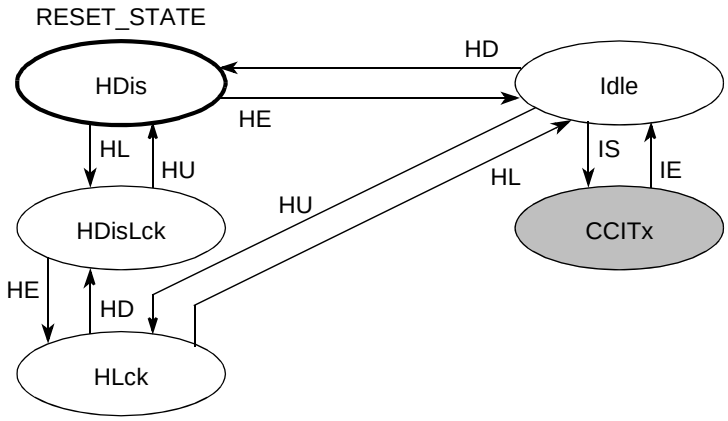


Figure 370. Double transmit message buffer state diagram (commit side)

A description of the states of the commit side of a double transmit message buffer is given in *Table 350*.

Table 350. Double transmit message buffer state description (commit side)

State	MBCCSR(2n)		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
CCITx	1	0	—	ITX	**Internal Message Transfer** — Message Buffer Data transferred from commit side to transmit side.
commit side specific states					
Idle	1	0	—	ITX, SS	**Idle** — Message Buffer Commit Side is idle. Commit Side can be used for internal message transfer.
HDisLck	0	1	CFG	SS	**Disabled and Locked** — Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
HLck	1	1	MSG	SS	Locked — Applications access to data, control, and status. Commit Side can *not* be used for internal message transfer.

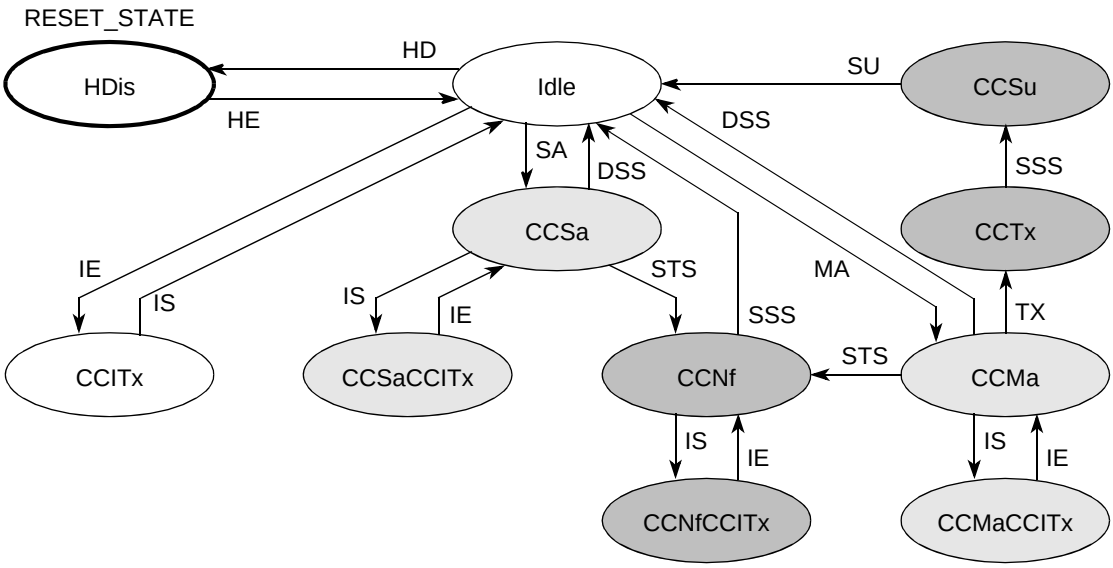


Figure 371. Double transmit message buffer state diagram (transmit side)

A description of the states of the transmit side of a double transmit message buffer is given in *Table 351*.

Table 351. Double transmit message buffer state description (transmit side)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Common states					
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
CCITx	1	0	—	TX	**Internal Message Transfer** — Message Buffer Data transferred from commit side to transmit side.
Transmit side specific states					
Idle	1	0	—	SR	**Idle** — Message Buffer Transmit Side is idle. Transmit Side is included in message buffer search.
CCSa	1	0	—	—	Slot Assigned — Message buffer assigned to next static slot. Ready for Null Frame transmission.
CCSaCCITx	1	0	—	TX	Slot Assigned and Internal Message Transfer — Message buffer assigned to next static slot and Message Buffer Data transferred from commit side to transmit side.
CCNf	1	0	—	TX	Null Frame Transmission Header is used for null frame transmission.
CCNfCCITx	1	0	—	TX	Null Frame Transmission and Internal Message Transfer — Header is used for null frame transmission and Message Buffer Data transferred from commit side to transmit side.

Table 351. Double transmit message buffer state description (transmit side) (continued)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCMa	1	0	—	—	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
CCMaCCITx	1	0	—	—	Message Available and Internal Message Transfer — Message buffer is assigned to next slot and cycle counter filter matches and Message Buffer Data transferred from commit side to transmit side.
CCTx	1	0	—	TX	Message Transmission — Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	—	SS	Status Update — Message buffer status update. Update of status flags, the slot status field, and the header index. Note: The slot status field of the commit side is updated too, even if the application has locked the commit side.

Message buffer transitions

0.0.0.0.0.19 Application transitions

The application transitions that can be triggered by the application using the commands described in *Table 352*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands can be issued on the transmit side only. Any enable or disable command issued on the commit side will be ignored without notification. The transitions that will be triggered depends on the value of the EDS bit. The enable and disable commands will affect both the commit side and the transmit side at the same time. If the application triggers the disable transition HD while the transmit side is in one of the states CCSa, CCSaCCITx, CCNf, CCNfCCITx, CCMa, CCMaCCITx, CCTx, or CCSu, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands can be issued on the commit side only. Any lock or unlock command issued on the transmit side will be ignored and the double transmit buffer lock error flag DBL_EF in the *CHI Error Flag Register (CHIERFR)* will be set. The transitions that will be triggered depends on the current value of the LCKS bit. The lock and unlock commands will only affect the commit side. If the application triggers the lock transition HL while the commit side is in the state CCITx, the message buffer state will not be changed and the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (CHIERFR)* will be set.

Table 352. Double Transmit Message Buffer Host Transitions

Transition	Host Command	Condition	Description
HE	MBCCSR(2n+1)[EDT]:=1	MBCCSR(2n+1)[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSR(2n+1)[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSR(2n)[LCKT]:=1	MBCCSR(2n)[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSR(2n)[LCKS] = 1	Application triggers message buffer unlock.

0.0.0.0.0.20 Module Transitions

The module transitions that can be triggered by the controller are described in *Table 353*. The transitions C1 and C2 apply to both sides of the message buffer and are applied at the same time. All other controller transitions apply to the transmit side only.

Table 353. Double Transmit Message Buffer Module Transitions

Transition	Condition	Description
common transitions		
IS	see *Section : Internal message transfer"*	Internal Message Transfer Start — Start transfer of message data from commit side to transmit side.
IE		Internal Message Transfer End — Stop transfer of message data from commit side to transmit side. Note: The internal message transfer is stopped before the slot or segment start.
transmit side specific transitions		
SA	slot match and static slot	Slot Assigned — Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSR(2n+1)[CMT]=1	Transmission Slot Start — Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated — Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start — Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start — Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start — Start of static slot or dynamic slot or symbol window or NIT.

0.0.0.0.0.21 Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 354*, the module transitions have a higher priority than the application transitions. The priorities among the controller transitions and the related states are given in the second part of *Table 354*. These priorities apply only to the transmit side. The internal message transmit start transition IS has tho lowest priority.

Table 354. Double transmit message buffer transition priorities

State	Priority	Description
Module vs. Application		
Idle	IS > HD IS > HL	Internal Message Transfer Start > Message Buffer Disable Internal Message Transfer Start > Message Buffer Lock
Module internal		
Idle	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

Message preparation

The application provides the message data through the commit side. The transmission itself is executed from the transmit side. The transfer of the message data from the commit side to the transmit side is done by the *Internal Message Transfer*, which is described in *Section : Internal message transfer.*

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field of the commit side and sets the commit bit CMT in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. The physical access to the message buffer data field is described in *Section : Individual message buffers.*

As indicated by *Table 350*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, or HLck. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 352*. The state change is indicated through the MBCCSRn[EDS] and MBCCSRn[LCKS] status bits.

Internal message transfer

The internal message transfer transfers the message data from the commit side to the transmit side. The internal message transfer is implemented as the swapping of the content of the *Message Buffer Index Registers (MBIDXRn)* of the commit side and the transmit side. After the swapping, the commit side CMT bit is cleared, the commit side interrupt flag MBIF is set, the transmit side CMT bit is set, and the transmit side DVAL bit is cleared.

The conditions and the point in time when the internal message transfer is started are controlled by the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. The MCM bit configures the message buffer for either the streaming commit mode or the immediate commit mode. A detailed description is given in *Streaming commit mode* and *Immediate commit mode*. The Internal Message Transfer is triggered with the transition IS. Both sides of the message buffer enter one of the CCITx states. The internal message transfer is finished with the transition IE.

0.0.0.0.0.22 Streaming commit mode

The intention of the streaming commit mode is to ensure that each committed message is transmitted *at least once*. The controller will not start the Internal Message Transfer for a message buffer as long as the message data on the transmit side is not transmitted at least once.

The streaming commit mode is configured by clearing the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for a double transmit message buffer when all of the following conditions are fulfilled:

1. the commit side is in the Idle state
   - the commit site message data are valid, i.e., MBCCSR(2*n*)[CMT] = 1
   - the transmit side is in one of the states Idle, CCSa, or CCMa
   - the transmit side contains either no valid message data, i.e., MBCCSR(2*n* + 1)[CMT] = 0 or the message data were transmitted at least once, i.e., MBCCSR(2*n* + 1)[DVAL] = 1

An example of a streaming commit mode state change diagram is given in *Figure 372*. In this example, both the commit and the transmit side do not contain valid message data and the application provides two messages. The message buffer does not match the next slot.

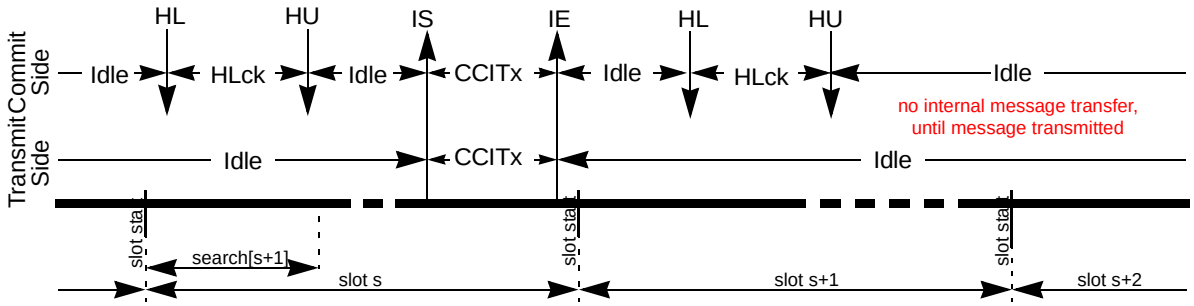


Figure 372. Internal message transfer in streaming commit mode

0.0.0.0.0.23 Immediate commit mode

The intention of the immediate commit mode is to transmit the *latest* data provided by the application. This implies that it is not guaranteed that each provided message will be transmitted at least once.

The immediate commit mode is configured by setting the message buffer commit mode bit MCM in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for one double transmit message buffer when all of the following conditions are fulfilled:

1. the commit side is in the Idle state
   - the commit site message data are valid, i.e., MBCCSR(2*n*)[CMT] = 1
   - the transmit side is in one of the states Idle, CCSa, or CCMa

It is not checked whether the transmit side contains no valid message data or valid message data were transmitted at least once. If message data are valid and not transmitted, they may be overwritten.

An example of a streaming commit mode state change diagram is given in *Figure 373*. In this example, both the commit and the transmit side do not contain valid message data, and the application provides two messages and the first message is gets overwritten. The message buffer does not match the next slot.

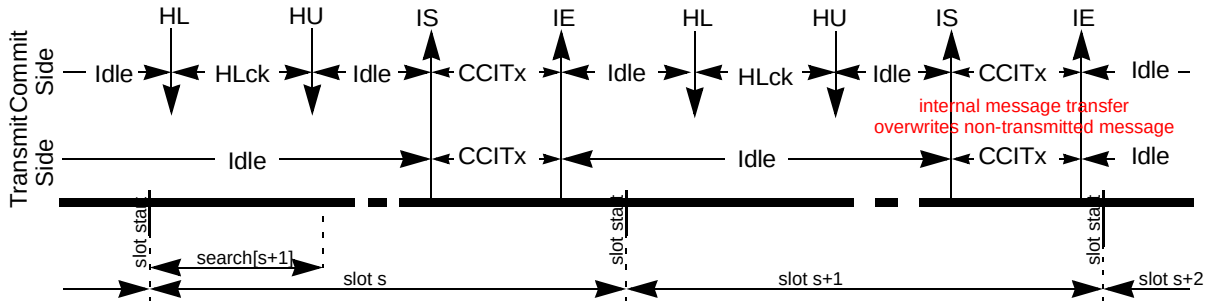


Figure 373. Internal message transfer in immediate commit mode

Message transmission

For double transmit message buffers, the message buffer search checks only the transmit side part. The internal scheduling ensures that the internal message transfer is stopped on the message buffer search start. Thus, the transmit side of message buffer, that is not in its transmission or status update slot, is always in the Idle state.

The message transmit behavior and transmission state changes of the transmit side of a double transmit message buffer are the same as for single buffered transmit buffers, except that the transmit side of double buffers cannot be locked by the application, i.e., the HU and HL transition do not exist. Therefore, refer to *Section : Message transmission.*

Message buffer status update

The message buffer status update behavior of the transmit side of a double transmit message buffer is the same as for single transmit message buffers which is described in *Section : Message buffer status update.*

Additionally, the slot status field of the commit side is update after the update of the slot status field of the transmit side, even if the commit side is locked by the application. This is implemented to provide the slot status of the most recent transmission slot.

21.6.7 Individual message buffer search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
   - slot start in the static segment
   - minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n+1*.

In general, the message buffer search for the next slot *n* considers only message buffers that are

1. Enabled, i.e., MBCCSR*n*[EDS] = 1, and
   - Matches the next slot *n*, i.e., MBFIDR*n*[FID] = *n*, and
   - Are the transmit side buffer in case of a double transmit message buffer.

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of *Table 355*. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of *Table 356*. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to *Table 355* for the static segment and according to *Table 356* for the dynamic segment are selected.

Table 355. Message buffer search priority (static segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	1	—	0	1	transmit buffer, matches cycle count, not committed	SA
	1	1	—	1	transmit buffer, matches cycle count, locked	SA
2	1	—	—	—	transmit buffer	SA
3	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section : Message buffer cycle counter filtering.*

Table 356. Message buffer search priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section : Message buffer cycle counter filtering.*

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in *Section : Message buffer channel assignment consistency.*

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in *Section : Individual message buffers.*

Message buffer cycle counter filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the *Message Buffer Cycle Counter Filter Registers (MBCCFRn)*. This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled, i.e., CCFE = 0, this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

Equation 29 ЗCCFRn[CCFE] =

Equation 30
Г & MBCCFRn[CCFMSK] = MBCCFRn[CCFVAL] & MBCCFRn[C

Message buffer channel assignment consistency

The message buffer channel assignment given by the CHA and CHB bits in the *Message Buffer Cycle Counter Filter Registers (MBCCFRn)* defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if MBCCFR*n*[CHA] = 1 and transmits or receives on channel B if MBCCFR*n*[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is shown in *Figure 374*.

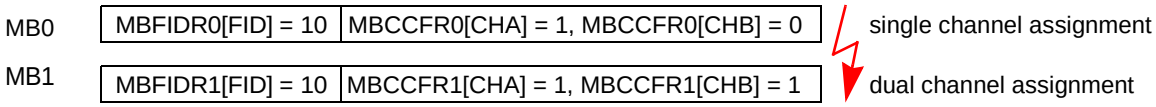


Figure 374. Inconsistent channel assignment

Node related slot multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to *Table 356* the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

Message buffer search error

If the message buffer search is running while the next message buffer search start event appears, the message buffer search is stopped and the Message Buffer Search Error Flag MSB_EF is set in the *CHI Error Flag Register (CHIERFR)*. This appears only if the CHI frequency is too low to search through all message buffers within the NIT or a minislot. The message buffer result is not defined in this case. For more details see *Section 21.7.3: Number of usable message buffers.*

21.6.8 Individual message buffer reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on *Specific configuration data*. The common configuration data given in the section on *Specific configuration data* cannot be reconfigured when the protocol is out of the *POC:config* state.

Reconfiguration schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

Basic type not changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if both the message buffer transfer direction bit MBCCSn[MTD] and the message buffer type bit MBCCSn[MBT] are not changed. This type of reconfiguration is denoted by RC1 in *Figure 375*. Single transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state. Double transmit message buffers can be RC1-reconfigured if both the transmit side and the commit side are in the HDis state.

Buffer type not changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer if the message buffer buffer type bit MBCCSRn[MBT] is not changed. This type of reconfiguration is denoted by RC2 in *Figure 375*. It applies only to single transmit and receive message buffers. Single transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

Buffer type changed (RC3)

A reconfiguration will change the buffer type of the individual message buffer if the message buffer type bit MBCCSRn[MBT] is changed. This type of reconfiguration is denoted by RC3 in *Figure 375*. The RC3 reconfiguration splits one double buffer into two single buffers or combines two single buffer into one double buffer. In the later case, the two single message buffers must have consecutive message buffer numbers and the smaller one must be even. Message Buffers can be RC3 reconfigured if they are in the HDis state.

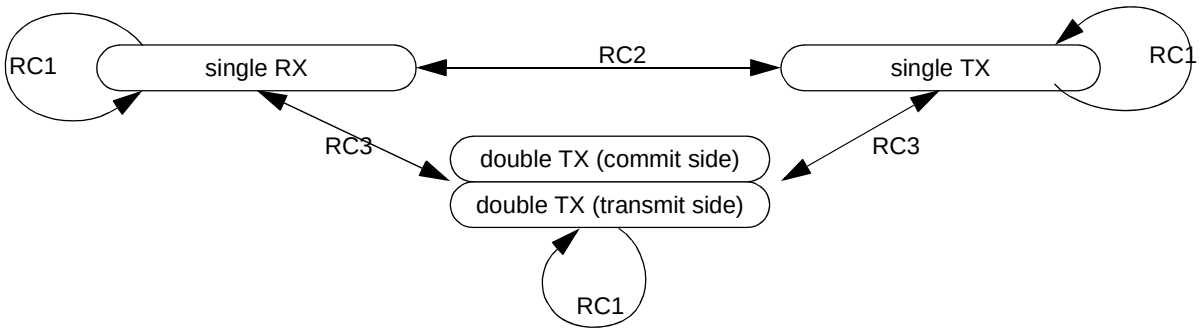


Figure 375. Message buffer reconfiguration scheme

21.6.9 Receive FIFO

This section provides a detailed description of the two receive FIFOs.

Overview

The receive FIFOs implement the queued receive buffer defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.* One receive FIFO is assigned to channel A, the other receive FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in *Section : Receive FIFO.* The area in the FlexRay memory for each of the two receive FIFOs is characterized by:

- The index of the first FIFO entry given by *Receive FIFO Start Index Register (RFSIR)*
- The number of FIFO entries and the length of each FIFO entry as given by *Receive FIFO Depth and Size Register (RFDSR)*

Receive FIFO configuration

The receive FIFO control and configuration data are given in *Section : Receive FIFO control and configuration data.* The configuration of the receive FIFOs consists of two steps.

The first step is the allocation of the required amount of FlexRay memory for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see *Section 21.6.4: FlexRay memory layout.*

The second step is the programming of the configuration data register while the PE is in *POC:config.*

The following steps configure the layout of the FIFO.

- The number of the first message buffer header index that belongs to the FIFO is written into the *Receive FIFO Start Index Register (RFSIR)*.
- The depth of the FIFO is written into the FIFO_DEPTH field in the *Receive FIFO Depth and Size Register (RFDSR)*.
- The length of the message buffer data field for the FIFO is written into the ENTRY_SIZE field in the *Receive FIFO Depth and Size Register (RFDSR)*.

Note: To ensure that the read index RDIDX always points to a message buffer that contains valid data, the receive FIFO must have at least two entries.

The FIFO filters are configured through the FIFO filter registers.

Receive FIFO reception

The frame reception to the receive FIFO is enabled if, for a certain slot, no message buffer is assigned or subscribed. In this case the FIFO filter path shown in *Figure 376* is activated.

When the receive FIFO filter path indicates that the received frame must be appended to the FIFO, the controller writes the received frame header and slot status into the message buffer header field indicated by the internal FIFO header write index. The payload data are written in the message buffer data field. If the status of the received frame indicates a valid frame, the internal FIFO header write index is updated and the FIFO not-empty interrupt flag FNEAIF/FNEBIF in the *Global Interrupt Flag and Enable Register (GIFER)* is set.

Receive FIFO message access

If the FIFO not-empty interrupt flag FNEAIF/FNEBIF in the *Global Interrupt Flag and Enable Register (GIFER)* is set, the receive FIFO contains valid received messages, which can be accessed by the application.

The receive FIFO does not require locking to access the message buffers. To access the message the application first reads the receive FIFO read index RDIDX from the *Receive FIFO A Read Index Register (RFARIR)* or *Receive FIFO B Read Index Register (RFBRIR)*, respectively. This index points to the message buffer header field of the next message buffer that contains valid data. The application can access the message data as described in *Section : Receive FIFO.* When the application has read all message buffer data and status information, it writes 1 to the FIFO not-empty interrupt flags FNEAIF or FNEBIF. This clears the interrupt flag and updates the RDIDX field in the *Receive FIFO A Read Index Register (RFARIR)* or *Receive FIFO B Read Index Register (RFBRIR)*, respectively.When the RDIDX value has reached the last message buffer header field that belongs to the FIFO, it wraps around to the index of the first message buffer header field that belongs to the FIFO. This value is provided by the SIDX field in the *Receive FIFO Start Index Register (RFSIR)*.

Receive FIFO filtering

The receive FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The controller provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is shown in *Figure 376*.

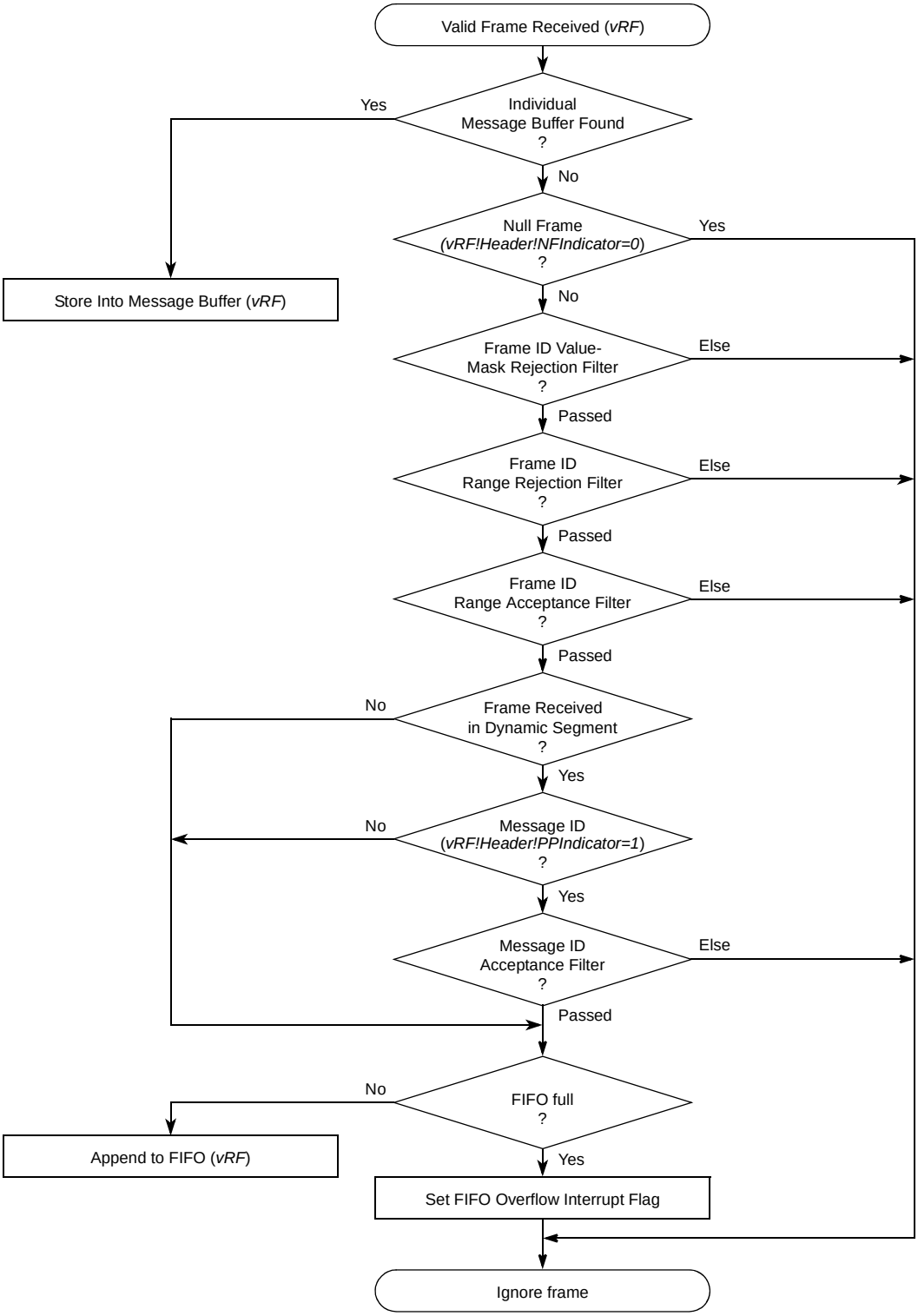

Valid Frame Received (vRF)
Individual Message Buffer Found ?
Yes
No
Store Into Message Buffer (vRF)
Null Frame (vRF!Header!NFIndicator=0) ?
Yes
No
Frame ID Value-Mask Rejection Filter ?
Else
Passed
Frame ID Range Rejection Filter ?
Else
Passed
Frame ID Range Acceptance Filter ?
Else
Passed
Frame Received in Dynamic Segment ?
No
Yes
Message ID (vRF!Header!PPIndicator=1) ?
No
Yes
Message ID Acceptance Filter ?
Else
Passed
FIFO full ?
No
Append to FIFO (vRF)
Yes
Set FIFO Overflow Interrupt Flag
Ignore frame

Figure 376. Received frame FIFO filter path

A received frame passes the FIFO filtering if it has passed all three types of filter.

RX FIFO frame ID value-mask rejection filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the *Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)* and the *Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)*. Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e., is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if *Equation 31* is fulfilled.

Equation 31

$$\text{FIDRFMR[FIDRFMSK]} \neq \text{RFFIDRFVR[FIDRFVAL]} \ \&\ \text{RFFIDRFMR[FID}$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- RFFIDRFVR[FIDRFVAL] := 0x000 and RFFIDRFMR[FIDRFMSK] := 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- RFFIDRFMR[FIDRFMSK] := 0x000

Using the settings above, *Equation 31* can never be fulfilled (0!= 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

RX FIFO frame ID range rejection filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the *Receive FIFO Range Filter Configuration Register (RFRFCFR)* and controlled by the *Receive FIFO Range Filter Control Register (RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e., RFRFCTR.FiMD = 1 and RFRFCTR.FiEN = 1, *Equation 32* is fulfilled.

Equation 32

$$(\text{FRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}=0}]) \text{ or } (\text{RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}=1}$$

Consequently, all frames with a frame ID that fulfills *Equation 33* for at least one of the enabled rejection filters will be rejected and thus not pass.

Equation 33

$$\text{FR}_{\text{SEL}}[\text{SID}_{\text{IBD}=0}] \leq \text{FID} \leq \text{RFRFCFR}_{\text{SEL}}[\text{SID}$$

RX FIFO frame ID range acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the *Receive FIFO Range Filter Configuration Register (RFRFCFR)* and controlled by the *Receive FIFO Range Filter Control Register (RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range

acceptance filters, i.e., RFRFCTR.FiMD = 0 and RFRFCTR.FiEN = 1, *Equation 34* is fulfilled.

Equation 34

$$RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq RFRFCFR_{SEL}[SID$$

RX FIFO message ID acceptance filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the *Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)* and the *Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)*. This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if *Equation 35* is fulfilled.

Equation 35

$$RFMIDAFMR[MIDAFMSK] = RFMIDAFMR[MIDAFVAL] \& RFMIDAFMR[MIDA$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- RFMIDAFMR[MIDAFMSK] := 0x000

Using the settings above, *Equation 35* is always fulfilled and all frames will pass.

21.6.10 Channel device modes

This section describes the two FlexRay channel device modes that are supported by the controller.

Dual channel device mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX, and FR_A_TX_EN is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and FR_B_TX_EN is connected to the physical bus channel B. The dual channel system is shown in *Figure 377*.

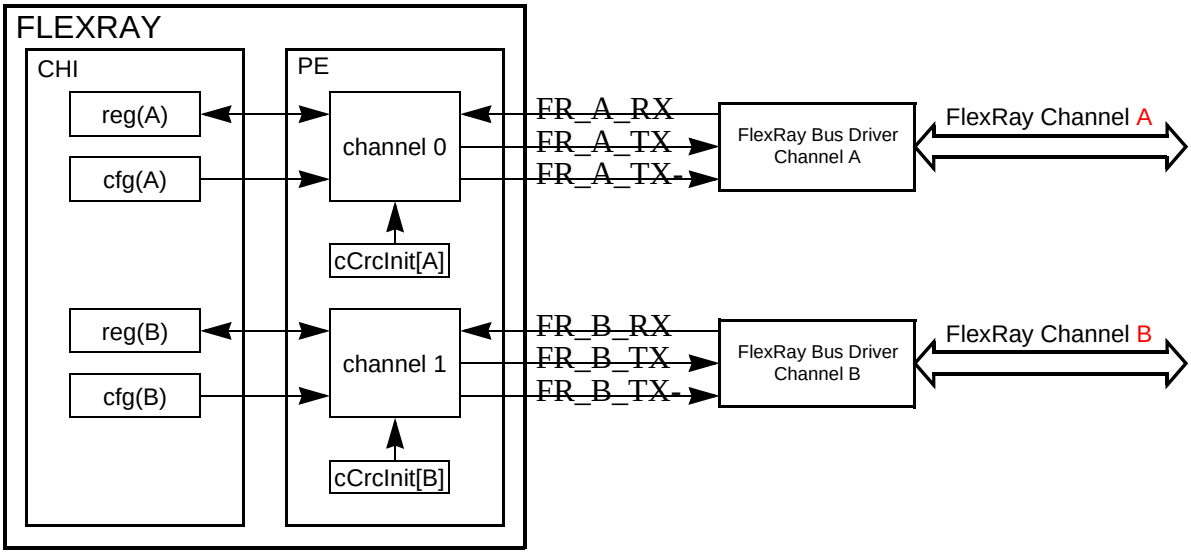


Figure 377. Dual channel device mode

Single channel device mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and FR_A_TX_EN and can be connected to either the physical bus channel A (shown in *Figure 378*) or the physical bus channel B (shown in *Figure 379*).

If the device is configured as a single channel device by setting MCR.SCD to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of MCR.CHA and MCR.CHB, the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit MCR.CHA must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit MCR.CHB must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

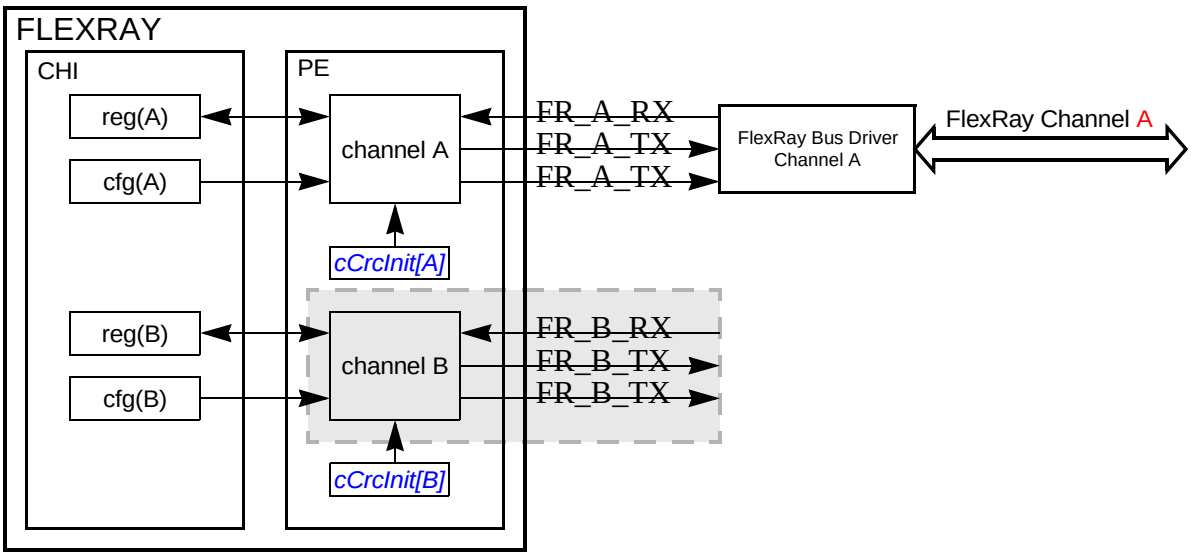


Figure 378. Single channel device mode (Channel A)

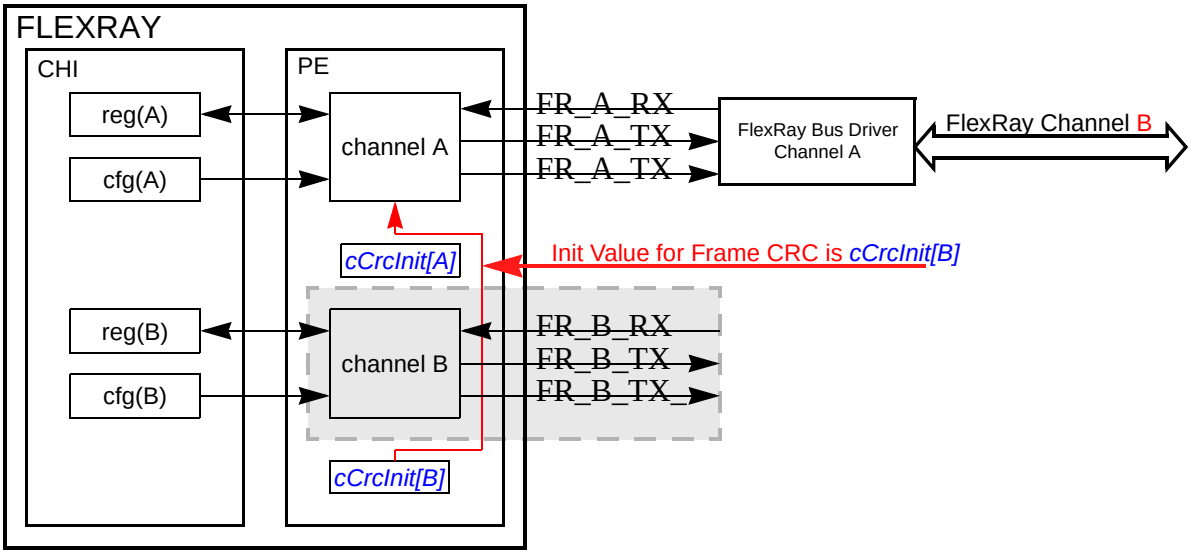


Figure 379. Single channel device mode (Channel B)

21.6.11 External clock synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the *Protocol Operation Control Register (POCR)*. The PE applies the external correction values in the next even-odd cycle pair as shown in *Figure 380* and *Figure 381*.

Note: The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2n+1 shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle 2n and before the end of the static segment of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle 2n+1. If the value is not applied in cycle 2n+1, then the value will be applied in the cycle 2n+3. Refer to *Figure 380* for timing details.

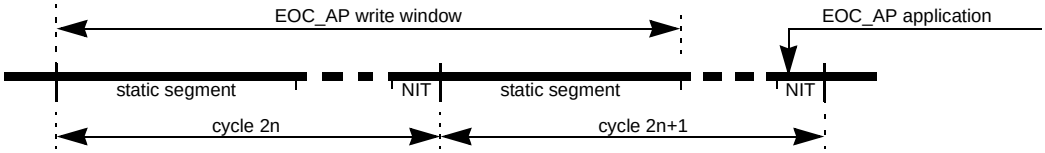


Figure 380. External offset correction write and application timing

If the rate correction for the cycle pair [2n+2, 2n+3] shall be affected by the external offset correction, the ERC_AP field must be written to after the start of cycle 2n and before the end of the static segment start of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle pair [2n+2, 2n+3]. If the value is not applied for cycle pair [2n+2, 2n+3], then the value will be applied for cycle pair [2n+4, 2n+5]. Refer to *Figure 381* for details.

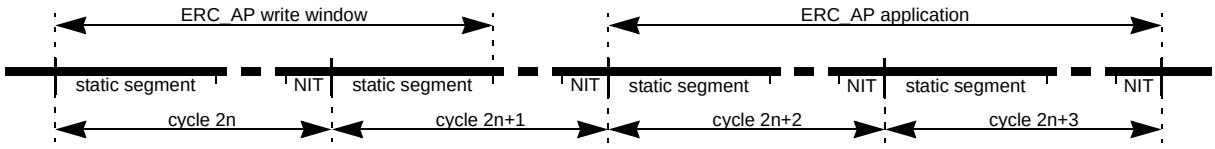


Figure 381. External rate correction write and application timing

21.6.12 Sync frame ID and sync frame deviation tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The controller provides the means to write a copy of these internal tables into the FlexRay memory and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the controller will not overwrite these tables and the application can read a consistent snapshot.

Note: Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

Sync frame ID table content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

Sync frame deviation table content

The Sync Frame Deviation Table is a snapshot of the protocol related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

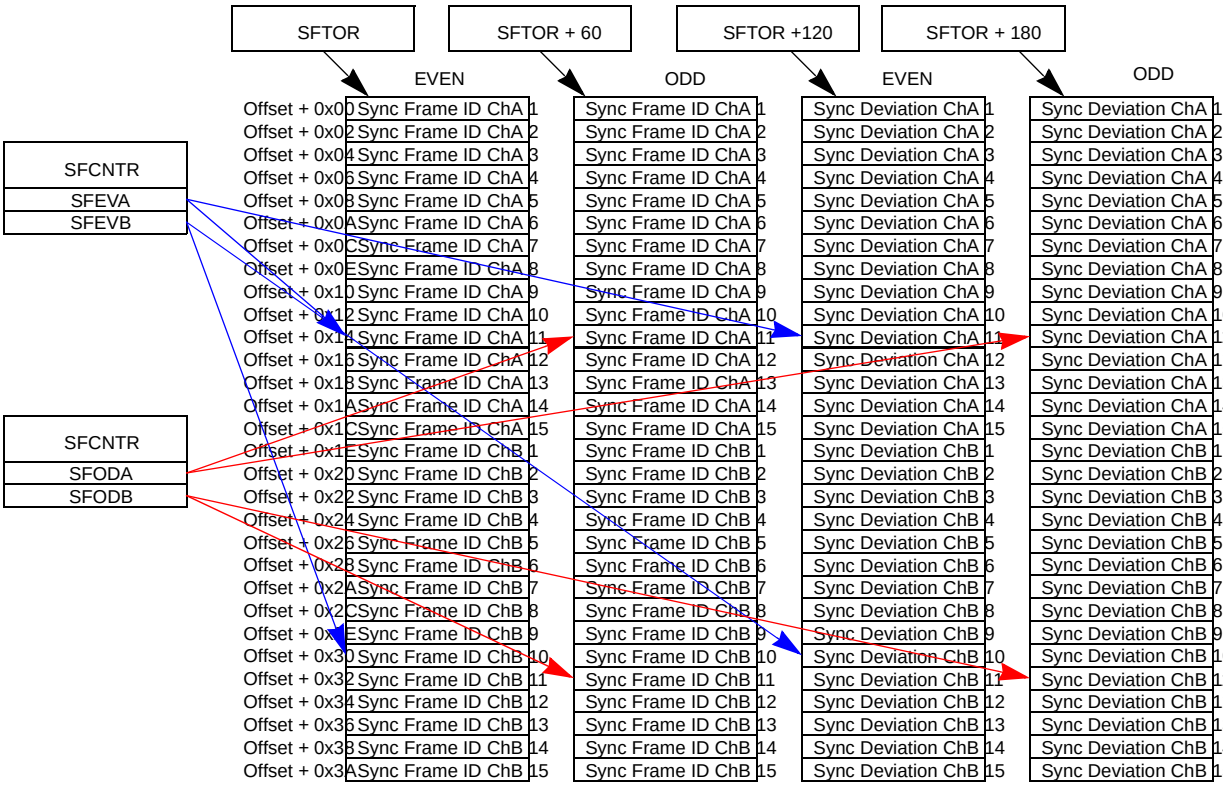


Figure 382. Sync table memory layout

Sync frame ID and sync frame deviation table setup

The controller writes a copy of the internal synchronization frame ID and deviation tables into the FlexRay memory if requested by the application. The application must provide the appropriate amount of FlexRay memory for the tables. The memory layout of the tables is given in *Figure 382*. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the *Sync Frame Table Offset Register (SFTOR)*.

Sync frame ID and sync frame deviation table generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the FlexRay memory using the *Sync Frame Table Configuration, Control, Status Register (SFTCCSR)*. A summary of the copy modes is given in *Table 357*.

Table 357. Sync frame table generation modes

SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
0	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting SFTCCSR.SIDEN to 1, the controller starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The controller checks if the application has locked the tables by reading the SFTCCSR.ELKS lock status bit. If this bit is set, the controller will not update the table in this cycle. If this bit is cleared, the controller locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the controller clears the even table valid status bit SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the FlexRay memory, the controller sets the even table valid bit SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the *Protocol Interrupt Flag Register 1 (PIFR1)*. If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the controller from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger SFTCCSR.ELKT. When the even table is not currently updated by the controller, the lock is granted and the even table lock status bit SFTCCSR.ELKS is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the *Sync Frame Counter Register (SFCNTR)*. The value in the SFTCCSR.CYCNUM field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the SFCNTR.SFEVA and SFCNTR.SFEVB fields. The application can now start to read the sync table data from the locations given in *Figure 382*.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger SFTCCSR.ELKT again. The even table lock status bit SFTCCSR.ELKS is reset immediately.

If the sync frame table generation is disabled, the table valid bits SFTCCSR[EVAL] and SFTCCSR[EVAL] are reset when the counter values in the *Sync Frame Counter Register (SFCNTR)* are updated. This is done because the tables stored in the FlexRay memory are no longer related to the values in the *Sync Frame Counter Register (SFCNTR)*.

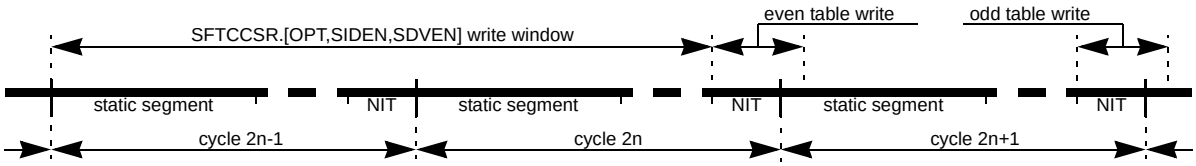


Figure 383. Sync frame table trigger and generation timing

Sync Frame Table Access

The sync frame tables will be transferred into the FlexRay memory during the table write windows shown in *Figure 383*. During the table write, the application cannot lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

Sync frame table locking and unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the *Sync Frame Table Configuration, Control, Status Register (SFTCCSR)*. If the affected table is not currently written to the FlexRay memory, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the FlexRay memory, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

21.6.13 MTS generation

The controller provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the *MTS A Configuration Register (MTSACFR)* and *MTS B Configuration Register (MTSBCFR)*.

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the *MTS A Configuration Register (MTSACFR)* or *MTS B Configuration Register (MTSBCFR)*. If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if *Equation 37*, *Equation 38*, and *Equation 38* are fulfilled.

Equation 360[PROTSTATE] = *POC:normal a*

Equation 37TSACRF[MTE] =

Equation 38
& MTSACFR[CYCCNTMSK] = MTSACFR[CYCCNTVAL] & MTSACFR[CYC

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if *Equation 36*, *Equation 39*, and *Equation 40* are fulfilled.

Equation 39 TSBCRF[MTE] =

Equation 40
& MTSBCFR[CYCCNTMSK] = MTSBCFR[CYCCNTVAL] & MTSBCFR[CYC(

21.6.14 Key slot transmission

Key slot assignment

A key slot is assigned to the controller if the key_slot_id field in the *Protocol Configuration Register 18 (PCR18)* is configured with a value greater than 0 and less or equal to number_of_static_slots in *Protocol Configuration Register 2 (PCR2)*, otherwise no key slot is assigned.

Key slot transmission in *POC:startup*

If a key slot is assigned and the controller is in the *POC:startup* state, startup null frames will be transmitted as specified by *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Key slot transmission in *POC:normal active*

If a key slot is assigned and the controller is in *POC:normal active*, a frame of the type as shown in *Table 358* is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see *Section : Message transmission"*). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see *Section : Null frame transmission"*).

Table 358. Key slot frame type

PCR11[key_slot_used_for_sync]	PCR11[key_slot_used_for_startup]	Key slot frame type
0	0	normal frame
0	1	normal frame(1)
1	0	sync frame
1	1	startup frame

1. The frame transmitted has an semantically incorrect header and will be detected as an invalid frame at the receiver.

21.6.15 Sync frame filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, i.e., the SFFE control bit in the *Module Configuration Register (MCR)* is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

Sync frame acceptance filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the *Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)* and the mask is configured in the *Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)*.

A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if *Equation 41* or *Equation 42* evaluates to true.

Equation 41 CR[SFFE] =

Equation 42 SFIDAFMR[FMSK] = SFIDAFVR[FVAL] & SFIDAFMR[

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

Sync frame rejection filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the *Sync Frame ID Rejection Filter Register (SFIDRFR)*. A received synchronization frame with the frame ID FID passes the sync frame rejection filter if *Equation 43* or *Equation 44* evaluates to true.

Equation 43 CR[SFFE] =

Equation 44 D ≠ SFIDRFR[SYNFRI

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

21.6.16 Strobe signal support

The controller provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in *Table 258*.

Strobe signal assignment

Each of the strobe signals listed in *Table 258* can be assigned to one of the four strobe ports using the *Strobe Signal Control Register (STBSCR)*. To assign multiple strobe signals, the application must write multiple times to the *Strobe Signal Control Register (STBSCR)* with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence:

1. Write to STBSCR with WMD = 1 and SEL = N. (updates SEL field only)
   – Read STBCSR.
   The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

Strobe signal timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in *Table 258* with a negative clock offset. An example waveform is given in *Figure 384*.

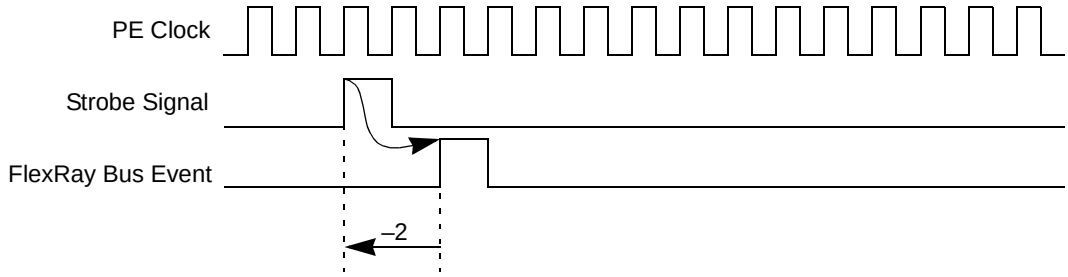


Figure 384. Strobe signal timing (type = pulse, clk_offset = –2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in *Table 258* with a positive clock offset. An example waveform is given in *Figure 385*.

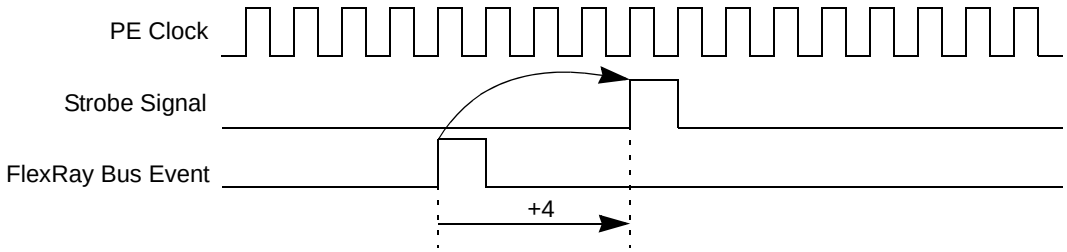


Figure 385. Strobe signal timing (type = pulse, clk_offset = +4)

21.6.17 Timer support

The controller provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

Absolute timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the *Protocol Interrupt Flag Register 0 (PIFR0)* is set at the macrotick start event, if *Equation* and *Equation 46* are fulfilled

Equation 45

CCNT] & TI1CYSR[T1_CYC_MSK] = TI1CYSR[T1_CYC_VAL] & TI1CYSR[T1_(

Equation 46

TR[MTCT] = TI1MTOR[T1_MTOFFS

If the timer 1 interrupt enable bit TI1_IE in the *Protocol Interrupt Enable Register 0 (PIER0)* is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

Absolute / relative timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the *Timer Configuration and Control Register (TICCR)*. The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

Absolute timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from *Timer 2 Configuration Register 0 (TI2CR0)* and *Timer 2 Configuration Register 1 (TI2CR1)* is used. On expiration of timer T2, the interrupt flag TI2_IF in the *Protocol Interrupt Flag Register 0 (PIFR0)* is set. If the timer 1 interrupt enable bit TI1_IE in the *Protocol Interrupt Enable Register 0 (PIER0)* is asserted, an interrupt request is generated.

Relative timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the *Protocol Interrupt Flag Register 0 (PIFR0)* is set, when the programmed amount of macroticks MT[31:0], defined by *Timer 2 Configuration Register 0 (TI2CR0)* and *Timer 2 Configuration Register 1 (TI2CR1)*, has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

21.6.18 Slot status monitoring

The controller provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is described in *Table 359*. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in *Figure 386*.

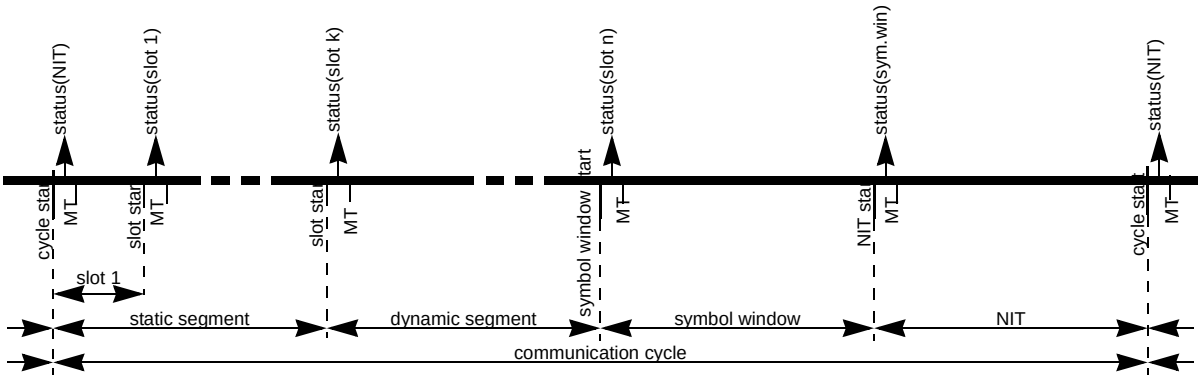


Figure 386. Slot status vector update

Note: The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 359. Slot status content

	Status Content

static / dynamic Slot	slot related status
	vSS!ValidFrame - valid frame received
	vSS!SyntaxError - syntax error occurred while receiving
	vSS!ContentError - content error occurred while receiving
	vSS!BViolation - boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict - reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict - reception ongoing while transmission starts
	first valid - channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if vSS!ValidFrame = 1
	b) last received header, if vSS!ValidFrame = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator - Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator - Startup Frame Indicator
	vRF!Header!SyFIndicator - Sync Frame Indicator

Symbol Window	window related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* - valid Media Test Access Symbol received received frame related status see static/dynamic slot
NIT	NIT related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 received frame related status see static/dynamic slot

Channel status error counter registers

The two channel status error counter registers, *Channel A Status Error Counter Register (CASERCR)* and *Channel B Status Error Counter Register (CBSERCR)*, incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*,

vSS!BViolation, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

Protocol status registers

The *Protocol Status Register 2 (PSR2)* provides slot status information about the Network Idle Time NIT and the Symbol Window. The *Protocol Status Register 3 (PSR3)* provides aggregated slot status information.

Slot status registers

The eight slot status registers, *Slot Status Registers (SSR0–SSR7)*, can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in *Table 359*, except of the *first valid* indicator for non-transmission slots.

Slot status counter registers

The controller provides four slot status error counter registers, *Slot Status Counter Registers (SSCR0–SSCR3)*. Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the *Slot Status Counter Condition Register (SSCCR)*, matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in *Figure 387*.

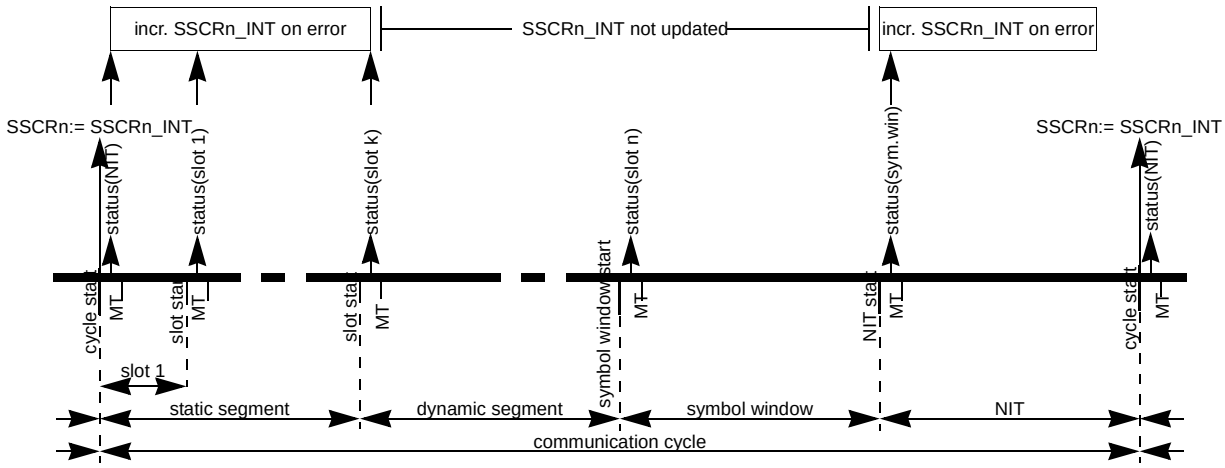


Figure 387. Slot status counting and SSCR*n* update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the SSCRn register reflects, in cycle n, the status of the NIT of cycle n-2, and the status of all static slots and the symbol window of cycle n-1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (SSCCR*n*.VFR | SSCCR*n*.SYF | SSCCR*n*.NUF | SSCCR*n*.SUF) // count on frame condition = 1;

and

- ((~SSCCR*n*.VFR | *vSS!ValidFrame*) & // valid frame restriction
  (~SSCCR*n*.SYF | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
  (~SSCCR*n*.NUF | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
  (~SSCCR*n*.SUF | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction = 1;

Note: The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

– slot related condition:

- ((SSCCR*n*.STATUSMASK[3] & *vSS!ContentError*) | // increment on content error
  (SSCCR*n*.STATUSMASK[2] & *vSS!SyntaxError*) | // increment on syntax error
  (SSCCR*n*.STATUSMASK[1] & *vSS!BViolation*) | // increment on boundary violation
  (SSCCR*n*.STATUSMASK[0] & *vSS!TxConflict*)) // increment on transmission conflict = 1;

If the slot status counter is in single cycle mode, i.e., SSCCR*n*[MCY] = 0, the internal slot status counter SSCR*n*_INT is reset at each cycle start. If the slot status counter is in the multicycle mode, i.e., SSCCR*n*[MCY] = 1, the counter is not reset and incremented, until the maximum value is reached.

Message buffer slot status field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in *Table 359* for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in *Section 21.6.6: Individual message buffer functional description.*

21.6.19 System bus access

This section provides a description of the system bus accesses performed by the controller.

All FlexRay memory data located in the system memory are accessed via the system bus. There are two types of failures that can occur during the system bus access, the system bus illegal address access and the system bus access timeout.

The behavior of the controller after the occurrence of a system bus failure is defined by the SBFF bit in the *Module Configuration Register (MCR)*.

System bus illegal address access

If the system bus detects an controller access to an illegal address, the controller receives a notification from the system bus about this event and sets the ILSA_EF flag in the *CHI Error Flag Register (CHIERFR)*.

System bus access timeout

The controller starts a timer when it has send an access request to the system bus. This timer expires after 2 × SYMATOR.TIMEOUT + 2 system bus clock cycles. If the access is not finished within this amount of time, the SBCF_EF flag in the *CHI Error Flag Register (CHIERFR)* is set.

Continue after system bus failure

If the SBFF bit in the *Module Configuration Register (MCR)* is 0, the controller will continue its operation after the occurrence of the system bus access failure but will not generate any system bus accesses until the start of the next communication cycle.

If a frame is under transmission when the system bus failure occurs, a correct frame is generated with the remaining header and frame data are replaced by all zeros. Depending on the point in time this can affect the PPI bit, the Header CRC, the Payload Length in case of an dynamic slot, and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of next communication cycle.

Freeze after system bus failure

If the SBFF bit in the *Module Configuration Register (MCR)* is set to 1, the controller will go into the freeze mode immediately after the occurrence of one of the system bus access failures.

21.6.20 Interrupt support

The controller provides 8 interrupt sources, out of them 5 are combined interrupt sources.

Individual interrupt sources

Receive FIFO interrupts

The controller provides 2 Receive FIFO interrupt sources.

Each of the 2 Receive FIFO provides a Receive FIFO Not Empty Interrupt Flag. The controller sets the Receive FIFO Not Empty Interrupt Flags (GIFER.FNEBIF, GIFER.FNEAIF) in the *Global Interrupt Flag and Enable Register (GIFER)* if the corresponding Receive FIFO is not empty.

Wakeup interrupt

The controller provides one interrupt source related to the wakeup.

The controller sets the Wakeup Interrupt Flag GIFER.WUPIF when it has received a wakeup symbol on the FlexRay bus. The controller generates an interrupt request if the interrupt enable bit GIFER.WUPIE is asserted.

Combined interrupt sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

Receive message buffer interrupt

The combined receive message buffer interrupt request RBIRQ is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.RBIE is set.

Transmit message buffer interrupt

The combined transmit message buffer interrupt request TBIRQ is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.TBIE is asserted.

Protocol interrupt

The controller provides 26 interrupt sources for protocol related events. For details, see *Protocol Interrupt Flag Register 0 (PIFR0)* and *Protocol Interrupt Flag Register 1 (PIFR1)*. Each interrupt source has its own interrupt enable bit.

The combined protocol interrupt request PRTIRQ is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit GIFER.PRIE is set.

Module interrupt

The combined module interrupt request MIRQ is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit GIFER.MIE is set.

CHI error interrupts

The controller provides 16 interrupt sources for CHI related error events. For details, see *CHI Error Flag Register (CHIERFR)*. There is one common interrupt enable bit GIFER.CHIIE for all CHI error interrupt sources.

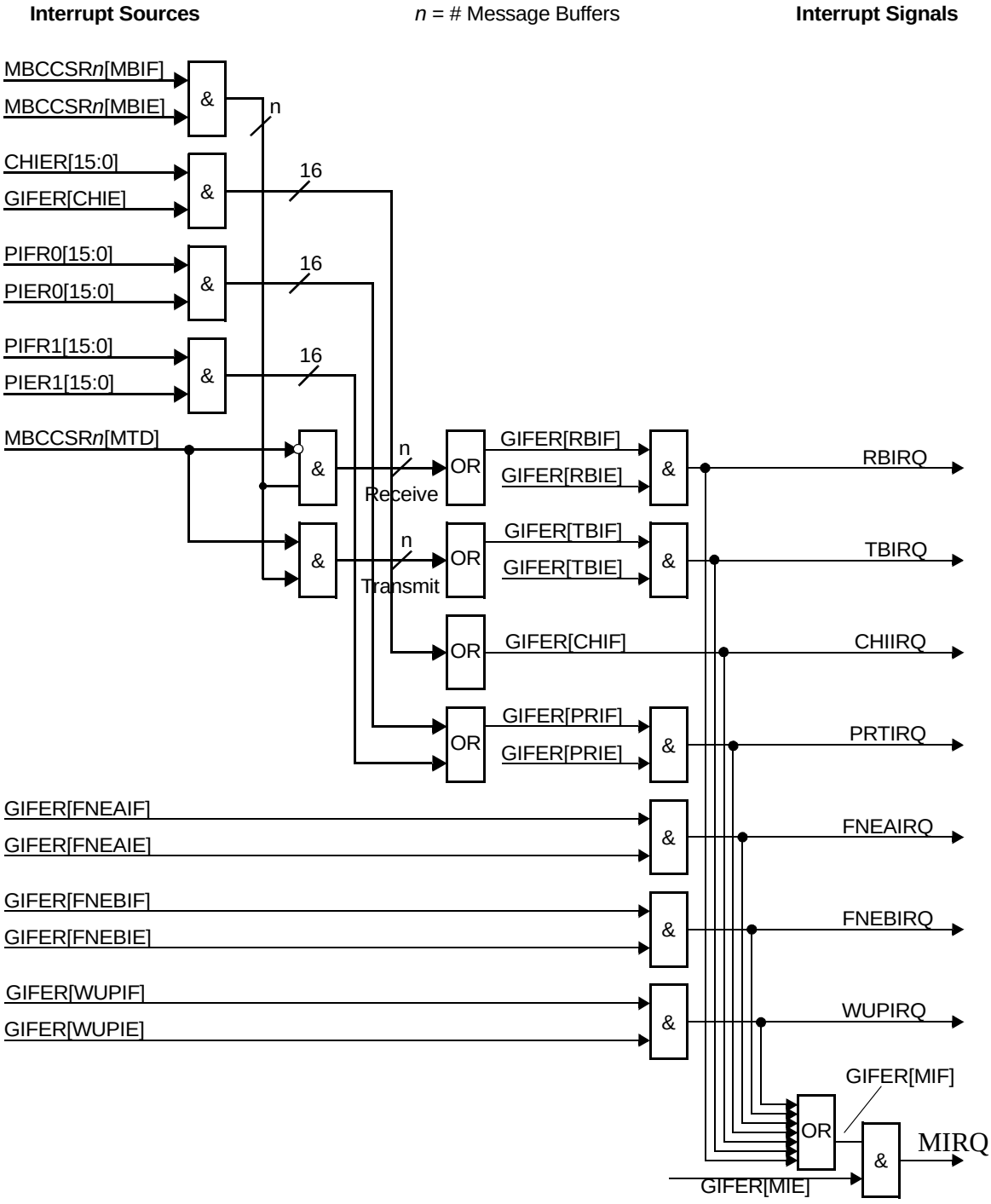


Figure 388. Scheme of cascaded interrupt request

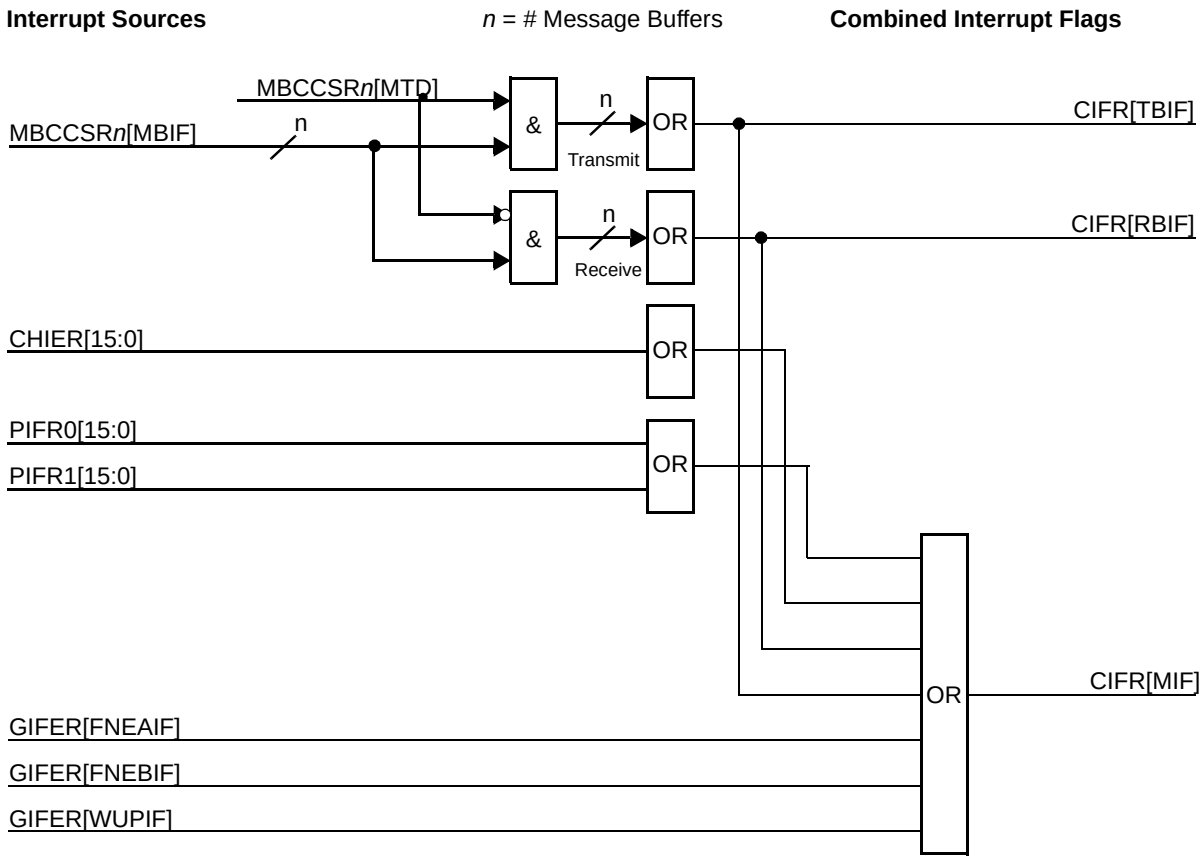


Figure 389. Scheme of combined interrupt flags

21.6.21 Lower bit rate support

The controller supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset.* The application configures the FlexRay channel bit rate by setting the BITRATE field in the *Module Configuration Register (MCR)*. The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in *Table 360*.

Table 360. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	MCR.BITRATE	*pdMicrotick* [ns]	*gdSampleClockPeriod* [ns]	pSamplesPerMicrotick	cSamplesPerBit	cStrobeOffset
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

Note: The bit rate of 8 Mbit/s is not defined by the FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

21.7 Application information

21.7.1 Initialization sequence

This section describes the required steps to initialize the controller. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the controller.

Module initialization

This section describes the module related initialization steps after a system reset.

1. Configure controller.
   a) configure the control bits in the *Module Configuration Register (MCR)*
   b) configure the system memory base address in *System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)*
   – Enable the controller.
   a) write 1 to the module enable bit MEN in the *Module Configuration Register (MCR)*

The controller now enters the Normal Mode. The application can commence with the protocol initialization described in *Section : Protocol initialization.*

Protocol initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
   a) issue CONFIG command via *Protocol Operation Control Register (POCR)*
   b) wait for *POC:config* in *Protocol Status Register 0 (PSR0)*
   c) configure the PCR0,..., PCR30 registers to set all protocol parameters
   – Configure the Message Buffers and FIFOs.
   a) set the number of message buffers used and the message buffer segmentation in the *Message Buffer Segment Size and Utilization Register (MBSSUTR)*
   b) define the message buffer data size in the *Message Buffer Data Size Register (MBDSR)*
   c) configure each message buffer by setting the configuration values in the *Message Buffer Configuration, Control, Status Registers (MBCCSRn)*, *Message Buffer Cycle Counter Filter Registers (MBCCFRn)*, *Message Buffer Frame ID Registers (MBFIDRn)*, *Message Buffer Index Registers (MBIDXRn)*
   d) configure the receive FIFOs
   e) issue CONFIG_COMPLETE command via *Protocol Operation Control Register (POCR)*
   f) wait for *POC:ready* in *Protocol Status Register 0 (PSR0)*

After this sequence, the controller is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

21.7.2 Shut down sequence

This section describes a secure shut down sequence to stop the controller gracefully. The main targets of this sequence are

- finish all ongoing reception and transmission
- do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a graceful shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) repeatedly write 1 to MBCCSRn[EDT] until MBCCSRn[EDS] == 0.
   – Stop Protocol Engine.
   a) issue HALT command via *Protocol Operation Control Register (POCR)*
   b) wait for *POC:halt* in *Protocol Status Register 0 (PSR0)*

21.7.3 Number of usable message buffers

This section describes the relationship between the number of message buffers that can be utilized and the required minimum CHI clock frequency. Additional constraints for the minimum CHI clock frequency are given in *Section 21.3: Controller host interface clocking.*

The controller uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search must be finished within one FlexRay slot. The shortest FlexRay slot is an empty dynamic slot. An empty dynamic slot is a minislot and consists of *gdMinislot* macroticks with a nominal duration of *gdMacrotick*. The minimum duration of a corrected macrotick is $gdMacrotick_{min}$ = 39 µT. This results in a minimum slot length of

Equation 47 $_{tmin}$ = 39 · *pdMicrotick* · *gdMini*

The search engine is located in the CHI and runs on the CHI clock. It evaluates one individual message buffer per CHI clock cycle. For internal status update and double buffer commit operations, and as a result of the clock domain crossing jitter, an additional amount of 10 CHI clock cycles is required to ensure correct operation. For a given number of message buffers and for a given CHI clock frequency f_{chi}, this results in a search duration of

Equation 48 $_{rch} = \frac{1}{f_{chi}} \cdot (\#\ \text{MessageBuffers} +$

The message buffer search must be finished within one slot which requires that *Equation 49* must be fulfilled

Equation 49 $_{search} \leq \Delta_{slotm}$

This results in the formula given in *Equation 50* which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

Equation 50 $\geq \frac{\#\ \text{MessageBuffers} + 10}{39 \cdot pdMicrotick \cdot gdMinis}$

The minimum CHI frequency for a selected set of relevant protocol parameters is given in *Table 361*.

Table 361. Minimum f_{chi} [MHz] examples (32 message buffers)

pdMicrotick [ns]	gdMinislot					
	2	3	4	5	6	7
25.0	21.54	14.35	10.77	8.61	7.18	6.16
50.0	10.73	7.18	5.39	4.31	3.59	3.08

21.7.4 Protocol control command execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of *Table 261* by writing the command to the POCCMD field of the *Protocol Operation Control Register (POCR)*. As a result the controller sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in *Table 362*. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the controller asserts the illegal protocol command interrupt flag IPC_IF in the *Protocol Interrupt Flag Register 1 (PIFR1)*. The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by *Table 362*. If the application issues the RESET, FREEZE, or

READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 362. Protocol control command priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
RESET	(highest) 1	none	
FREEZE	2		RESET
READY	3		RESET
CONFIG_COMPLETE	3		RESET
ALL_SLOTS	4	RESET, FREEZE, READY, CONFIG_COMPLET, fatal protocol error	RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		RESET
RUN	6		RESET, FREEZE, fatal protocol error
WAKEUP	7		RESET, FREEZE, fatal protocol error
DEFAULT_CONFIG	8		RESET, FREEZE, fatal protocol error
CONFIG	9		RESET
HALT	(lowest) 10		RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

21.7.5 Protocol reset command

The application can use the RESET command to initiate a reset of the protocol engine

The application issues the protocol reset command by writing the RESET command to the POCCMD field of the *Protocol Operation Control Register (POCR)*.

As a result, the PE stops its operation within the next 500 ns. At the same time the FlexRay bus ports are put into the reset state given in *Table 247*. The protocol and message buffer configuration data are not affected by the RESET command and will not be changed.

The duration of the RESET command must be at least 500 ns.

Before accessing the message buffers, the application must put the protocol into the *POC:default config* state by sending the Protocol Command DEFAULT_CONFIG via *Protocol Operation Control Register (POCR)*.

21.7.6 Message buffer search on simple message buffer configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The receive FIFO behavior is not considered in this section.

Simple message buffer configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple

configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and has following configuration

Table 363. Transmit buffer configuration

Register	Field	Value	Description
MBCCSR*t*	MCM	—	used only for double buffers
	MBT	0	single transmit buffer
	MTD	1	transmit buffer
MBCCFR*t*	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = {4n} = {0,4,8,12,...}
	CCFVAL	000000	
MBFIDR*t*	FID	S	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit MBCCSRt[CMT] and the lock bit MBCCSRt[LCKS].

The receive message buffer has the message buffer number *r* and has following configuration

Table 364. Receive buffer configuration

Register	Field	Value	Description
MBCCSR*r*	MCM	—	n/a
	MBT	—	n/a
	MTD	0	receive buffer
MBCCF*r*	MTM	—	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6,...}
	CCFVAL	000000	
MBFIDR*r*	FID	S	subscribed slot

Furthermore the assumption is that both message buffers are enabled (MBCCSR*t*[EDS] = 1 and MBCCSR*r*[EDS] = 1)

Note: The cycle set {4n+2} = {2,6,10,...} is assigned to the receive buffer only.

The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

Behavior in static segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S.* In any case, the result of the message buffer search will be the transmit message buffer *t.* The receive message buffer *r* will not be found, no reception is possible.

Behavior in dynamic segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

Transmit data not available

If transmit data are *not available*, i.e., the transmit buffer is not committed MBCCSR*t*[CMT] = 0 and/or locked MBCCSR*t*[LCKS] = 1,

a) for the cycles in the set {4n}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) for the cycles in the set {4n+2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in *Figure 390*.

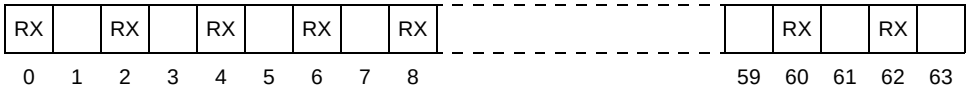


Figure 390. Transmit data not available

Transmit data available

If transmit data are *available,* i.e., the transmit buffer is committed MBCCSR*t*[CMT] = 1 and not locked MBCCSR*t*[LCKS] = 0,

a) for the cycles in the set {4*n*}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) for the cycles in the set {4*n* + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in *Figure 391*.

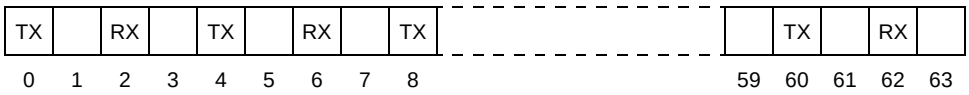


Figure 391. Transmit data not available

22 Deserial Serial Peripheral Interface (DSPI)

22.1 Introduction

This chapter describes the deserial serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

The SPC56xP60/54 implements the modules DSPI0 to 4. The “x” appended to signal names signifies the module to which the signal applies. Thus CS0_x specifies that the CS0 signal applies to DSPI module 0, 1, etc.

22.2 Block diagram

A block diagram of the DSPI is shown in *Figure 392*.

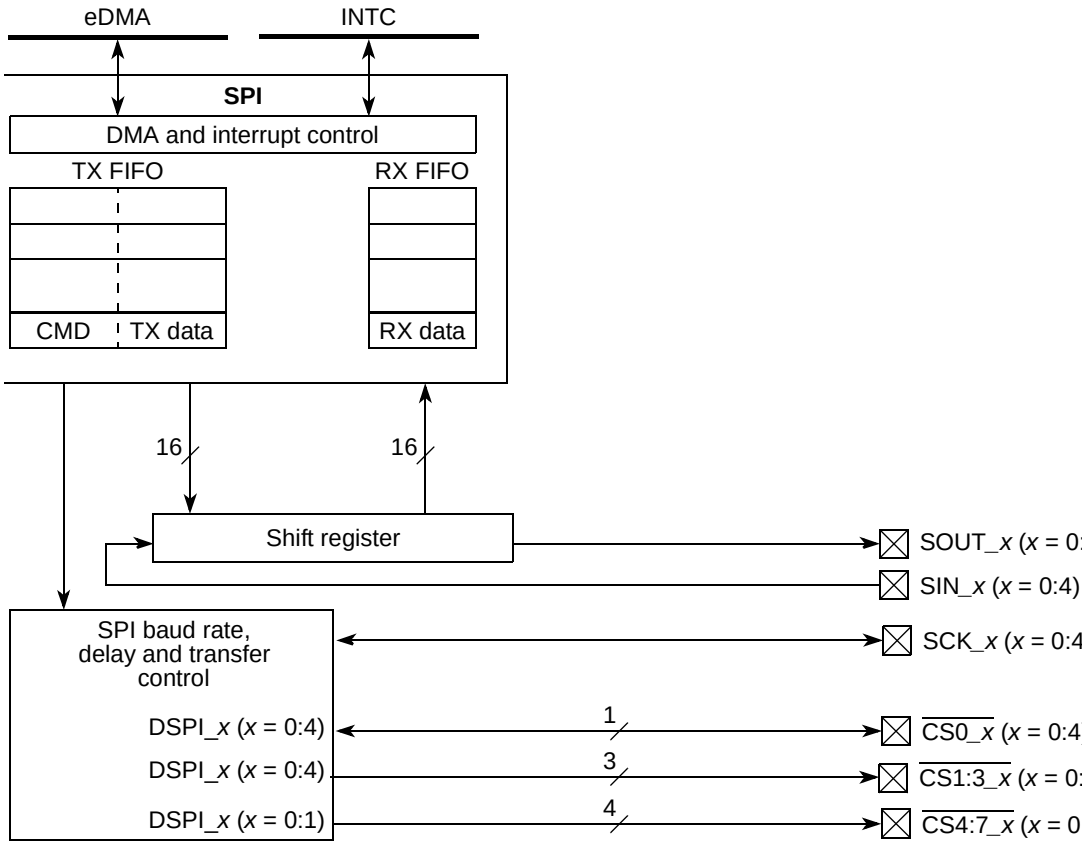


Figure 392. DSPI block diagram

22.3 Overview

The register content is transmitted using an SPI protocol. There are five DSPI modules (DSPI_0 – DSPI_4) on the device. The modules are identical except that DSPI_0 and DSPI_1 have four additional chip select ($\overline{CS}$) lines.

For queued operations, the SPI queues reside in internal SRAM that is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

Figure 393 shows a DSPI with external queues in internal SRAM.

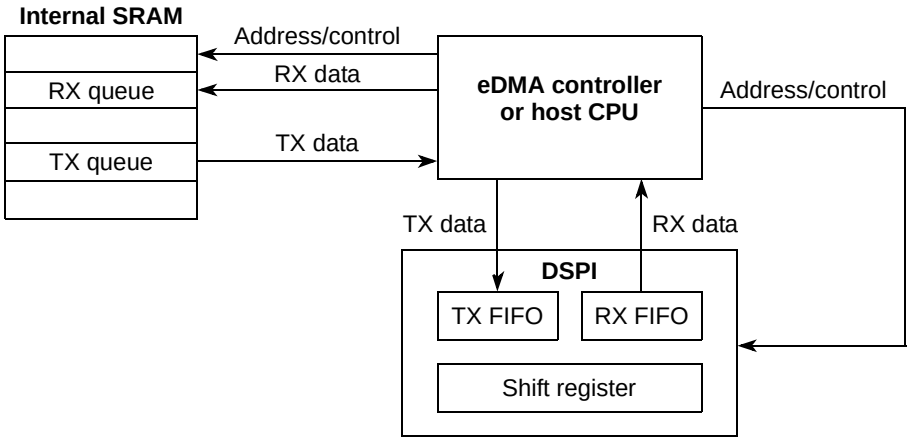


Figure 393. DSPI with queues and eDMA

22.4 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave modes
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 5 entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - 8 clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    CS to SCK delay
    SCK to CS delay
    Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- 8 peripheral chip selects, expandable up to 256 with external demultiplexer
- Deglitching support for as many as 32 peripheral chip selects with external demultiplexer
- 2 DMA conditions for SPI queues residing in RAM or flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - RX FIFO is not empty (RFDF)
  - FIFO overrun (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF)
  - FIFO under flow (slave only and SPI mode, the slave is asked to transfer data when the TX FIFO is empty) (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (SCK)

22.5 Modes of operation

The DSPI has four modes of operation. These modes can be divided into two categories; module-specific modes such as master, slave, and module disable modes, and a second category that is an MCU-specific mode: debug mode. All four modes are implemented on this device.

The module-specific modes are entered by host software writing to a register. The MCU-specific mode is controlled by signals external to the DSPI. The MCU-specific mode is a mode that the entire device may enter, in parallel to the DSPI being in one of its module-specific modes.

22.5.1 Master mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK, CS*n* and SOUT signals are controlled by the DSPI and configured as outputs.

For more information, refer to *Section : Master mode*.

22.5.2 Slave mode

Slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode. In slave mode, the SCK signal and the CS0_*x* signal are configured as inputs and provided by a bus master. CS0_*x* must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

For more information, refer to *Section : Slave mode*.

22.5.3 Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

For more information, refer to *Section : Module disable mode*.

22.5.4 Debug mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected.

For more information, refer to *Section : Debug mode*.

22.6 External signal description

22.6.1 Signal overview

Table 365 lists off-chip DSPI signals.

Table 365. Signal properties

Name	I/O type	Function	
		Master mode	Slave mode
CS0_*x*	Output / input	Peripheral chip select 0	Slave select
CS1:3_*x* (DSPI 0: CS1:3_0, CS5_0) (DSPI 1: CS1:3_1, CS5_1)	Output	Peripheral chip select 1–3	Unused(1)
CS4_x	Output	Peripheral chip select 4	Master trigger

Table 365. Signal properties (continued)

Name	I/O type	Function	
		Master mode	Slave mode
$\overline{CS5\_x}$ (DSPI 0: $\overline{CS7\_0}$) (DSPI 1: $\overline{CS7\_1}$)	Output	Peripheral chip select 5 / Peripheral chip select strobe	Unused(1)
SIN_*x*	Input	Serial data in	Serial data in
SOUT_*x*	Output	Serial data out	Serial data out
SCK_*x*	Output / input	Serial clock (output)	Serial clock (input)

1. The SIUL allows you to select alternate pin functions for the device.

22.6.2 Signal names and descriptions

Peripheral Chip Select / Slave Select (CS_0)

In master mode, the CS_0 signal is a peripheral chip select output that selects the slave device to which the current transmission is intended.

In slave mode, the CS_0 signal is a slave select input signal that allows an SPI master to select the DSPI as the target for transmission. CS_0 must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

Set the IBE and OBE bits in the PCR register for all CS_0 pins when the DSPI chip select or slave select primary function is selected for that pin. When the pin is used for DSPI master mode as a chip select output, set the OBE bit. When the pin is used in DSPI slave mode as a slave select input, set the IBE bit.

Refer to *Section : Pad Configuration Registers (PCR[0:107])* for more information.

Peripheral Chip Selects 1–3 ($\overline{CS1:3}$)

$\overline{CS1:3}$ are peripheral chip select output signals in master mode. In slave mode these signals are not used.

Peripheral Chip Select 4 ($\overline{CS4}$)

$\overline{CS4}$ is a peripheral chip select output signal in master mode.

Peripheral Chip Select 5/Peripheral Chip Select Strobe ($\overline{CS\_5}$)

$\overline{CS5}$ is a peripheral chip select output signal. When the DSPI is in master mode and PCSSE bit in the DSPI*x*_MCR is cleared, the $\overline{CS5}$ signal selects the slave device that receives the current transfer.

$\overline{CS5}$ is a strobe signal used by external logic for deglitching of the CS signals. When the DSPI is in master mode and the PCSSE bit in the DSPI*x*_MCR is set, the $\overline{CS5}$ signal indicates the timing to decode $\overline{CS0:4}$ signals, which prevents glitches from occurring.

$\overline{CS5}$ is not used in slave mode.

Serial Input (SIN_*x*)

SIN_*x* is a serial data input signal.

Serial Output (SOUT_*x*)

SOUT_*x* is a serial data output signal.

Serial Clock (SCK_*x*)

SCK_*x* is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK_*x* is an input from an external bus master.

22.7 Memory map and registers description

22.7.1 Memory map

Table 366 shows the DSPI memory map.

Table 366. DSPI memory map

Offset from DSPI_BASE 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2) 0xFFF9_C000 (DSPI_3) 0x8FFA_0000 (DSPI_4)	Register	Location
0x0000	DSPI_MCR—DSPI module configuration register	*on page 678*
0x0004	Reserved	
0x0008	DSPI_TCR—DSPI transfer count register	*on page 681*
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	*on page 682*
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	*on page 682*
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	*on page 682*
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	*on page 682*
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	*on page 682*
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	*on page 682*
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	*on page 682*
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	*on page 682*
0x002C	DSPI_SR—DSPI status register	*on page 688*

Table 366. DSPI memory map (continued)

Offset from DSPI_BASE 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2) 0xFFF9_C000 (DSPI_3) 0x8FFA_0000 (DSPI_4)	Register	Location
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	*on page 690*
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	*on page 691*
0x0038	DSPI_POPR—DSPI pop RX FIFO register	*on page 693*
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	*on page 694*
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	*on page 694*
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	*on page 694*
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	*on page 694*
0x004C	DSPI_TXFR4—DSPI transmit FIFO register 4	*on page 694*
0x0050–0x007B	Reserved	
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	*on page 695*
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	*on page 695*
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	*on page 695*
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	*on page 695*
0x008C	DSPI_RXFR4—DSPI receive FIFO register 4	*on page 695*
0x0090–0x3FFF	Reserved	

22.7.2 Registers description

DSPI Module Configuration Register (DSPI*x*_MCR)

The DSPI*x*_MCR contains bits that configure attributes of the DSPI operation. The values of the HALT and MDIS bits can be changed at any time, but their effect begins on the next frame boundary. The HALT and MDIS bits in the DSPI*x*_MCR are the only bit values software can change while the DSPI is running.

Figure 394. DSPI Module Configuration Register (DSPI*x*_MCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR	CONT_SCKE	DCONF[0:1]		FRZ	MTFE	PCSSE	ROOE	PCSIS7	PCSIS6	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MDIS	DIS_TXF	DIS_RXF	CLR_TXF	CLR_RXF	SMPL_PT[0:1]		0	0	0	0	0	0	0	HALT
W					w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 367. DSPI*x*_MCR field descriptions

Field	Description
0 MSTR	Master/slave mode select Configures the DSPI for master mode or slave mode. 0 DSPI is in slave mode. 1 DSPI is in master mode.
1 CONT_SCKE	Continuous SCK enable Enables the serial communication clock (SCK) to run continuously. Refer to *Section 22.8.6: Continuous Serial communications clock* for details. 0 Continuous SCK disabled 1 Continuous SCK enabled If the FIFO is enabled with continuous SCK mode, before setting the CONT_SCKE bit, the TX-FIFO should be cleared and only CTAR0 register should be used for transfer attributes otherwise a change in SCK frequency occurs.
2–3 DCONF [0:1]	DSPI configuration The following table lists the DCONF values for the various configurations. **DCONF** / **Configuration** 00 / SPI 01 / Invalid value 10 / Invalid value 11 / Invalid value
4 FRZ	Freeze Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers. 1 Halt serial transfers.

Table 367. DSPI*x*_MCR field descriptions (continued)

Field	Description
5 MTFE	Modified timing format enable Enables a modified transfer format to be used. Refer to *Section : Modified SPI transfer format (MTFE = 1, CPHA = 1)* for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
6 PCSSE	Peripheral chip select strobe enable Enables the $\overline{CS5\_x}$ to operate as a $\overline{CS}$ strobe output signal. Refer to *Section : Peripheral Chip Select strobe enable (CS5_x)* for more information. 0 $\overline{CS5\_x}$ is used as the Peripheral chip select 5 signal. 1 $\overline{CS5\_x}$ is used as an active-low $\overline{CS}$ strobe signal.
7 ROOE	Receive FIFO overflow overwrite enable Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored. Refer to *Section : Receive FIFO overflow interrupt request (RFOF)* for more information. 0 Incoming data is ignored. 1 Incoming data is put in the shift register.
8–9	Reserved, but implemented. These bits are writable, but have no effect.
10–15 PCSIS*n*	Peripheral chip select inactive state Determines the inactive state of the $\overline{CS0\_x}$ signal. $\overline{CS0\_x}$ must be configured as inactive high for slave mode operation. 0 The inactive state of $\overline{CS0\_x}$ is low. 1 The inactive state of $\overline{CS0\_x}$ is high. PCSIS7 and PSCIS6 are implemented only on DSPI_0.
16	Reserved
17 MDIS	Module disable Allows the clock to stop to the non-memory mapped logic in the DSPI, effectively putting the DSPI in a software controlled power-saving state. Refer to *Section 22.8.8: Power saving features* for more information." The reset value of the MDIS bit is parameterized, with a default reset value of 0. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
18 DIS_TXF	Disable transmit FIFO Enables and disables the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. Refer to *Section : FIFO disable operation* for details. 0TX FIFO enabled 1TX FIFO disabled
19 DIS_RXF	Disable receive FIFO Enables and disables the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. Refer to *Section : FIFO disable operation* for details. 0 RX FIFO enabled 1 RX FIFO disabled

Table 367. DSPI*x*_MCR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20
CLR_TXF</td><td>Clear TX FIFO
Flushes the TX FIFO. Write a 1 to the CLR_TXF bit to clear the TX FIFO counter. The CLR_TXF bit is always read as 0.
0 Do not clear the TX FIFO counter.
1 Clear the TX FIFO counter.</td></tr>
<tr><td>21
CLR_RXF</td><td>Clear RX FIFO
Flushes the RX FIFO. Write a 1 to the CLR_RXF bit to clear the RX counter. The CLR_RXF bit is always read as 0.
0 Do not clear the RX FIFO counter.
1 Clear the RX FIFO counter.</td></tr>
<tr><td>22–23
SMPL_PT
[0:1]</td><td>Sample point
Allows the host software to select when the DSPI master samples SIN in modified transfer format. Figure 409 shows where the master can sample the SIN pin. The following table lists the delayed sample points.
<table><tr><th>SMPL_PT</th><th>Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x</th></tr><tr><td>00</td><td>0</td></tr><tr><td>01</td><td>1</td></tr><tr><td>10</td><td>2</td></tr><tr><td>11</td><td>Invalid value</td></tr></table></td></tr>
<tr><td>24–30</td><td>Reserved</td></tr>
<tr><td>31
HALT</td><td>Halt
Provides a mechanism for software to start and stop DSPI transfers. Refer to Section 22.8.2: Start and stop of DSPI transfers for details on the operation of this bit.
0 Start transfers.
1 Stop transfers.</td></tr>
</table>

DSPI Transfer Count Register (DSPI*x*_TCR)

The DSPI*x*_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. The user must not write to the DSPI*x*_TCR while the DSPI is running.

Figure 395. DSPI Transfer Count Register (DSPIx_TCR)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SPI_TCNT[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 368. DSPI*x*_TCR field descriptions

Field	Description
0–15 SPI_TCNT [0:15]	SPI transfer counter Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter 'wraps around,' incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved

DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTAR*n*)

The DSPI modules each contain eight clock and transfer attribute registers (DSPI*x*_CTAR*n*) that define different transfer attribute configurations. Each DSPI*x*_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB or LSB first

DSPI*x*_CTARs support compatibility with the QSPI module used in certain members of the SPC56x family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPI*x*_CTAR that contains the transfer's attributes. Do not write to the DSPI*x*_CTARs while the DSPI is running.

In master mode, the DSPI*x*_CTAR*n* registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit fields in the DSPI*x*_CTAR0 and DSPI*x*_CTAR1 registers sets the slave transfer attributes. Refer to the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI*x*_CTAR registers is used on a per-frame basis. When the DSPI is configured as an SPI bus slave, the DSPI*x*_CTAR0 register is used.

Figure 396. DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn)

Address: Base + 0x000C (DSPIx_CTAR0), Base + 0x0010 (DSPIx_CTAR1), Base + 0x0014 (DSPIx_CTAR2), Base + 0x0018 (DSPIx_CTAR3), Base + 0x001C (DSPIx_CTAR4), Base + 0x0020 (DSPIx_CTAR5), Base + 0x0024 (DSPIx_CTAR6), Base + 0x0028 (DSPIx_CTAR7)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSB FE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 369. DSPIx_CTARn field descriptions

Field	Descriptions
0 DBR	Double Baud Rate The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in Master Mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in *Table 370*. See the BR[0:3] field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set. 0 The baud rate is computed normally with a 50/50 duty cycle. 1 The baud rate is doubled with the duty cycle depending on the baud rate prescaler.
1–4 FMSZ[0:3]	Frame Size The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. *Table 371* lists the frame size encodings.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low. 1 The inactive state value of SCK is high.
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock phase settings. Continuous SCK is only supported for CPHA = 1. 0 Data is captured on the leading edge of SCK and changed on the following edge. 1 Data is changed on the leading edge of SCK and captured on the following edge.

Table 369. DSPI*x*_CTARn field descriptions (continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>7
LSBFE</td><td>LSB First
The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in Master Mode.
0 Data is transferred MSB first.
1 Data is transferred LSB first.</td></tr>
<tr><td>8–9
PCSSCK[0:1]</td><td>PCS to SCK Delay Prescaler
The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in Master Mode. The table lists the prescaler values. See the CSSCK[0:3] field description for details on how to compute the PCS to SCK delay.
<table>
<tr><th>PCSSCK</th><th>PCS to SCK delay prescaler value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>10–11
PASC[0:1]</td><td>After SCK Delay Prescaler
The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. This field is only used in Master Mode. The table lists the prescaler values. See the ASC[0:3] field description for details on how to compute the After SCK delay.
<table>
<tr><th>PASC</th><th>After SCK delay prescaler value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>12–13
PDT[0:1]</td><td>Delay after Transfer Prescaler
The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in Master Mode. The table lists the prescaler values. See the DT[0:3] field description for details on how to compute the delay after transfer.
<table>
<tr><th>PDT</th><th>Delay after transfer prescaler value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
</table>

Table 369. DSPI*x*_CTARn field descriptions (continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>14–15
PBR[0:1]</td><td>Baud Rate Prescale
The PBR field selects the prescaler value for the baud rate. This field is only used in Master Mode. The baud rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The baud rate prescaler values are listed in the table. See the BR[0:3] field description for details on how to compute the baud rate.
<table>
<tr><th>PBR</th><th>Baud rate prescaler value</th></tr>
<tr><td>00</td><td>2</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>16–19
CSSCK[0:3]</td><td>PCS to SCK Delay Scaler
The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in Master Mode. The PCS to SCK delay is the delay between the assertion of PCS and the first edge of the SCK. Table 372 lists the scaler values. The PCS to SCK delay is a multiple of the system clock period and it is computed according to the following equation:
Equation 51
$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$
See Section : CS to SCK delay (t_{CSC}) for more details.</td></tr>
<tr><td>20–23
ASC[0:3]</td><td>After SCK Delay Scaler
The ASC field selects the scaler value for the After SCK delay. This field is only used in Master Mode. The After SCK delay is the delay between the last edge of SCK and the negation of PCS. Table 373 lists the scaler values. The After SCK delay is a multiple of the system clock period, and it is computed according to the following equation:
Equation 52
$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$
See Section : After SCK delay (t_{ASC}) for more details.</td></tr>
<tr><td>24–27
DT[0:3]</td><td>Delay after Transfer Scaler
The DT field selects the delay after transfer scaler. This field is only used in Master Mode. The delay after transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. Table 374 lists the scaler values. In continuous serial communications clock operation, the DT value is fixed to one TSCK. The delay after transfer is a multiple of the system clock period and it is computed according to the following equation:
Equation 53
$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$
See Section : Delay after transfer (tDT) for more details.</td></tr>
</table>

Table 369. DSPI*x*_CTARn field descriptions (continued)

Field	Descriptions
28–31 BR[0:3]	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the SCK. *Table 375* lists the baud rate scaler values. The baud rate is computed according to the following equation: **Equation 54** $$\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$$ See *Section : Baud rate generator* for more details.

Table 370. DSPI SCK duty cycle

DBR	CPHA	PBR	SCK duty cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

Table 371. DSPI transfer frame size

FMSZ	Frame size	FMSZ	Frame size
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

Table 372. DSPI PCS to SCK delay scaler

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to sck delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 373. DSPI after SCK delay scaler

ASC	After SCK delay scaler value	ASC	After SCK delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 374. DSPI delay after transfer scaler

DT	Delay after transfer scaler value	DT	Delay after transfer scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 375. DSPI baud rate scaler

BR	Baud rate scaler value	BR	Baud rate scaler value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

DSPI Status Register (DSPI*x*_SR)

The DSPI*x*_SR contains status and flag bits. The bits are set by the hardware and reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI*x*_SR by writing a 1 to clear it (w1c). Writing a 0 to a flag bit has no effect.

Figure 397. DSPI Status Register (DSPI*x*_SR)

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 376. DSPI*x*_SR field descriptions

Field	Description
0 TCF	Transfer complete flag Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section : Classic SPI transfer format (CPHA = 0)* for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete. 1 Transfer complete.
1 TXRXS	TX and RX status Reflects the status of the DSPI. Refer to *Section 22.8.2: Start and stop of DSPI transfers* for information on what clears and sets this bit. 0 TX and RX operations are disabled (DSPI is in STOPPED state). 1 TX and RX operations are enabled (DSPI is in RUNNING state).

Table 376. DSPI*x*_SR field descriptions (continued)

Field	Description
2	Reserved
3 EOQF	End of queue flag Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section : Classic SPI transfer format (CPHA = 0)* for details. The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing command. 1 EOQ bit is set in the executing SPI command. EOQF does not function in slave mode.
4 TFUF	Transmit FIFO underflow flag Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred. 1 TX FIFO underflow has occurred.
5	Reserved
6 TFFF	Transmit FIFO fill flag Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it, or an by acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full. 1 TX FIFO is not full.
7–11	Reserved
12 RFOF	Receive FIFO overflow flag Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred. 1 RX FIFO overflow has occurred.
13	Reserved
14 RFDF	Receive FIFO drain flag Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it, or by acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty. 1 RX FIFO is not empty. In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.
15	Reserved

Table 376. DSPI*x*_SR field descriptions (continued)

Field	Description
16–19 TXCTR [0:3]	TX FIFO counter Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
20–23 TXNXTPTR [0:3]	Transmit next pointer Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. Refer to *Section : Transmit First In First Out (TX FIFO) buffering mechanism* for more details.
24–27 RXCTR [0:3]	RX FIFO counter Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section : Classic SPI transfer format (CPHA = 0)* for details.
28–31 POPNXTPTR [0:3]	Pop next pointer Contains a pointer to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR is updated when the DSPI*x*_POPR is read. Refer to *Section : Receive First In First Out (RX FIFO) buffering mechanism* for more details.

DSPI DMA / Interrupt Request Select and Enable Register (DSPI*x*_RSER)

The DSPI*x*_RSER serves two purposes: enables flag bits in the DSPI*x*_SR to generate DMA requests or interrupt requests, and selects the type of request to generate. Refer to the bit descriptions for the type of requests that are supported. Do not write to the DSPI*x*_RSER while the DSPI is running.

Figure 398. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 377. DSPI*x*_RSER field descriptions

Field	Description
0 TCF_RE	Transmission complete request enable Enables TCF flag in the DSPI*x*_SR to generate an interrupt request. 0 TCF interrupt requests are disabled. 1 TCF interrupt requests are enabled.
1–2	Reserved

Table 377. DSPI*x*_RSER field descriptions (continued)

Field	Description
3 EOQF_RE	DSPI finished request enable Enables the EOQF flag in the DSPI*x*_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled. 1 EOQF interrupt requests are enabled.
4 TFUF_RE	Transmit FIFO underflow request enable The TFUF_RE bit enables the TFUF flag in the DSPI*x*_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled. 1 TFUF interrupt requests are enabled.
5	Reserved
6 TFFF_RE	Transmit FIFO fill request enable Enables the TFFF flag in the DSPI*x*_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled. 1 TFFF interrupt requests or DMA requests are enabled.
7 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI*x*_SR is set, and the TFFF_RE bit in the DSPI*x*_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected. 1 DMA request is selected.
8–11	Reserved
12 RFOF_RE	Receive FIFO overflow request enable Enables the RFOF flag in the DSPI*x*_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled. 1 RFOF interrupt requests are enabled.
13	Reserved
14 RFDF_RE	Receive FIFO drain request enable Enables the RFDF flag in the DSPI*x*_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled. 1 RFDF interrupt requests or DMA requests are enabled.
15 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI*x*_SR is set, and the RFDF_RE bit in the DSPI*x*_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected. 1 DMA request is selected.
16–31	Reserved

DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

The DSPI*x*_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. Refer to *Section : Transmit First In First Out (TX FIFO) buffering mechanism* for more information. Write accesses of 8 or 16 bits to the DSPI*x*_PUSHR transfer 32 bits to the TX FIFO.

Note: TXDATA is used in master and slave modes.

Figure 399. DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

Address Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CTCNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 378. DSPI*x*_PUSHR field descriptions

Field	Description
0 CONT	Continuous peripheral chip select enable Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected $\overline{CS}$ signals to remain asserted between transfers. Refer to *Section : Continuous selection format* for more information. 0 Return peripheral chip select signals to their inactive state between transfers. 1 Keep peripheral chip select signals asserted between transfers.
1–3 CTAS [0:2]	Clock and transfer attributes select Selects which of the DSPI*x*_CTARs sets the transfer attributes for the SPI frame. In SPI slave mode, DSPI*x*_CTAR0 is used. The following table shows how the CTAS values map to the DSPI*x*_CTARs. There are eight DSPI*x*_CTARs in the device DSPI implementation. Use in SPI master mode only.

CTAS	Use Clock and Transfer Attributes from
000	DSPI*x*_CTAR0
001	DSPI*x*_CTAR1
010	DSPI*x*_CTAR2
011	DSPI*x*_CTAR3
100	DSPI*x*_CTAR4
101	DSPI*x*_CTAR5
110	DSPI*x*_CTAR6
111	DSPI*x*_CTAR7

Table 378. DSPI*x*_PUSHR field descriptions (continued)

Field	Description
4 EOQ	End of queue Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI*x*_SR is set. 0 The SPI data is not the last data to transfer. 1 The SPI data is the last data to transfer. Use in SPI master mode only.
5 CTCNT	Clear SPI_TCNT Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPI*x*_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear SPI_TCNT field in the DSPI*x*_TCR. 1 Clear SPI_TCNT field in the DSPI*x*_TCR. Use in SPI master mode only.
6–7	Reserved
10–15 PCS*x*	Peripheral chip select *x* Selects which $\overline{\text{CS}}$*x* signals are asserted for the transfer. 0 Negate the $\overline{\text{CS}}$*x* signal. 1 Assert the $\overline{\text{CS}}$*x* signal. Use in SPI master mode only.
16–31 TXDATA [0:15]	Transmit data Holds SPI data for transfer according to the associated SPI command. Use TXDATA in master and slave modes.

DSPI POP RX FIFO Register (DSPI*x*_POPR)

The DSPI*x*_POPR allows you to read the RX FIFO. Refer to *Section : Receive First In First Out (RX FIFO) buffering mechanism* for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI*x*_POPR fetch the RX FIFO data, and update the counter and pointer.

Note: Reading the RX FIFO field fetches data from the RX FIFO. Once the RX FIFO is read, the read data pointer is moved to the next entry in the RX FIFO. Therefore, read DSPIx_POPR only when you need the data. For compatibility, configure the TLB (MMU table) entry for DSPIx_POPR as guarded.

Figure 400. DSPI POP RX FIFO Register (DSPI*x*_POPR)

Address: Base + 0x0038 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 379. DSPI*x*_POPR field descriptions

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [0:15]	Received data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNXTPTR).

DSPI Transmit FIFO Registers 0–4 (DSPI*x*_TXFR*n*)

The DSPI*x*_TXFR*n* registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI*x*_TXFR*n* registers does not alter the state of the TX FIFO. The MCU uses five registers to implement the TX FIFO, that is DSPI*x*_TXFR0–DSPI*x*_TXFR4 are used.

Figure 401. DSPI Transmit FIFO Register 0–4 (DSPI*x*_TXFR*n*)

Address: Base + 0x003C (DSPI*x*_TXFR0)
Base + 0x0040 (DSPI*x*_TXFR1)
Base + 0x0044 (DSPI*x*_TXFR2)
Base + 0x0048 (DSPI*x*_TXFR3)
Base + 0x004C (DSPI*x*_TXFR4)
Access: User read-only

	0	1	2	4	5	6	7	8	9	10	11	12	13	14	15
R	TXCMD														
W															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 380. DSPI*x*_TXFR*n* field descriptions

Field	Description
0–15 TXCMD [0:15]	Transmit command Contains the command that sets the transfer attributes for the SPI data. Refer to *Section : DSPI PUSH TX FIFO Register (DSPIx_PUSHR)* for details on the command field.
16–31 TXDATA [0:15]	Transmit data Contains the SPI data to be shifted out.

DSPI Receive FIFO Registers 0–4 (DSPI*x*_RXFR*n*)

The DSPI*x*_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI*x*_RXFR registers are read-only. Reading the DSPI*x*_RXFR*n* registers does not alter the state of the RX FIFO. The device uses five registers to implement the RX FIFO, that is DSPI*x*_RXFR0–DSPI*x*_RXFR4 are used.

Figure 402. DSPI Receive FIFO Registers 0–4 (DSPIx_RXFRn)

Address: Base + 0x007C (DSPI*x*_RXFR0), Base + 0x0080 (DSPI*x*_RXFR1), Base + 0x0084 (DSPI*x*_RXFR2), Base + 0x0088 (DSPI*x*_RXFR3), Base + 0x008C (DSPI*x*_RXFR4)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 381. DSPI*x*_RXFR*n* field description

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [15:0]	Receive data Contains the received SPI data.

22.8 Functional description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. All communications are through an SPI-like protocol.

The DSPI supports only the serial peripheral interface (SPI) configuration in which the DSPI operates as a basic SPI or a queued SPI.

The DCONF field in the DSPI*x*_MCR register determines the DSPI configuration. Refer to *Table 367* for the DSPI configuration values.

The DSPI*x*_CTAR0–DSPI*x*_CTAR7 registers hold clock and transfer attributes.The SPI configuration can select which CTAR to use on a frame by frame basis by setting the CTAS field in the DSPI*x*_PUSHR.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT_*x* and SIN_*x* signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master's shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, the TCF bit in the DSPI*x*_SR is set to indicate a completed transfer. *Figure 403* illustrates how master and slave data is exchanged.

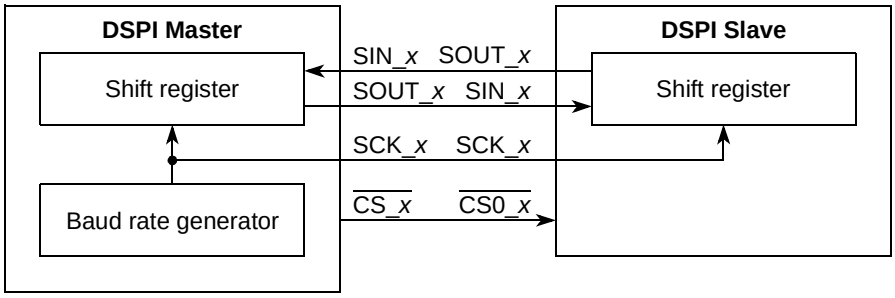


Figure 403. SPI serial protocol overview

Each DSPI has four peripheral chip select ($\overline{CSx}$) signals that select the slaves with which to communicate (DSPI_0 and DSPI_1 have eight CS*x* signals.)

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in *Section 22.8.5: Transfer formats.* The transfer rate and delay settings are described in *Section 22.8.4: DSPI baud rate and clock delay generation.*

Refer to *Section 22.8.8: Power saving features* for information on the power-saving features of the DSPI.

22.8.1 Modes of operation

The DSPI modules have four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode. All four modes are implemented on this device.

The module-specific modes are determined by bits in the DSPI*x*_MCR. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

Master mode

In master mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPI*x*_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI*x*_CTARs set the transfer attributes. Transfer attribute control is on a frame by frame basis.

Refer to *Section 22.8.3: Serial Peripheral Interface (SPI) configuration* for more details.

Slave mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI*x*_MCR is negated. The DSPI slave is selected by a bus master by having the slave's $\overline{CS0\_x}$ asserted. In slave mode, the SCK is provided by the bus master. All transfer attributes are controlled by the bus master, except the clock polarity, clock phase, and the number of bits to transfer. These must be configured in the DSPI slave for correct communications.

Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

Refer to *Section 22.8.8: Power saving features* for more details on the module disable mode.

Debug mode

The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller.

Refer to *Figure 404* for a state diagram.

22.8.2 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPI*x*_SR is cleared in this state. In the RUNNING state, serial transfers take place. The TXRXS bit in the DSPI*x*_SR is set in the RUNNING state.

Figure 404 shows a state diagram of the start and stop mechanism.

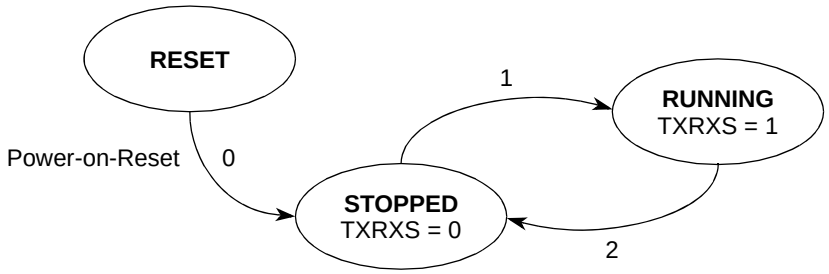


Figure 404. DSPI start and stop state diagram

The transitions are described in *Table 382*.

Table 382. State transitions for start and stop of DSPI transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI starts (transitions from STOPPED to RUNNING) when all of the following conditions are true: – EOQF bit is clear – Debug mode is unselected or the FRZ bit is clear – HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: – EOQF bit is set – Debug mode is selected and the FRZ bit is set – HALT bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

22.8.3 Serial Peripheral Interface (SPI) configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPI*x*_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI.

The FIFO buffer operations are described in *Section : Transmit First In First Out (TX FIFO) buffering mechanism* and *Section : Receive First In First Out (RX FIFO) buffering mechanism."*

The interrupt and DMA request conditions are described in *Section 22.8.7: Interrupts/DMA requests."*

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

SPI master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK_*x*) and the peripheral chip select ($\overline{\text{CS}}x$) signals. The SPI command field in the executing TX FIFO entry determines which CTARs set the transfer attributes and which $\overline{\text{CS}}x$ signals to assert. The command field also contains various bits that help with queue management and transfer protocol. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT_*x*) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

Refer to *Section : DSPI PUSH TX FIFO Register (DSPIx_PUSHR)* for details on the SPI command fields.

SPI slave mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPI*x*_CTAR0.

FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a 1 to the DIS_TXF bit in the DSPI*x*_MCR. The RX FIFO is disabled by writing a 1 to the DIS_RXF bit in the DSPI*x*_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI*x*_PUSHR and received data is read from the DSPI*x*_POPR. When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_TXFRs and TXNXTPTR are undefined. When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_RXFRs and POPNXTPTR are undefined.

Disable the TX and RX FIFOs only if the FIFO must be disabled as a requirement of the application's operating mode. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and can result in incorrect results.

Transmit First In First Out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds five entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI*x*_PUSHR). For more information on DSPI*x*_PUSHR refer to *Section : DSPI PUSH TX FIFO Register (DSPIx_PUSHR)."* TX FIFO entries can only be removed from the TX

FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPI*x*_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

Refer to *Section : DSPI Status Register (DSPIx_SR)* for more information on DSPI*x*_SR.

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI*x*_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI*x*_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI*x*_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPI*x*_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI*x*_PUSHR is complete or alternatively by host software writing a 1 to the TFFF in the DSPI*x*_SR. The TFFF can generate a DMA request or an interrupt request.

Refer to *Section : Transmit FIFO fill interrupt or DMA request (TFFF)* for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPI*x*_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a 1 to the CLR_TXF bit in DSPI*x*_MCR.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave's DSPI TX FIFO is empty, the transmit FIFO underflow flag (TFUF) in the slave's DSPI*x*_SR is set.

Refer to *Section : Transmit FIFO underflow interrupt request (TFUF)* for details.

Receive First In First Out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds five received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPI*x*_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI*x*_POPR or by flushing the RX FIFO.

Refer to *Section : DSPI POP RX FIFO Register (DSPIx_POPR)* for more information on the DSPI*x*_POPR.

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPI*x*_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPI*x*_SR points to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from

DSPI*x*_RXFR0. For example, POPNXTPTR equal to two means that the DSPI*x*_RXFR2 contains the received SPI data that is returned when DSPI*x*_POPR is read. The POPNXTPTR field is incremented every time the DSPI*x*_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI*x*_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored.

Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI*x*_POPR. A read of the DSPI*x*_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

Refer to *Section : DSPI POP RX FIFO Register (DSPIx_POPR)* for more information on DSPI*x*_POPR.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPI*x*_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPI*x*_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a 1 to it.

22.8.4 DSPI baud rate and clock delay generation

The SCK_*x* frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate.

Figure 405 shows conceptually how the SCK signal is generated.

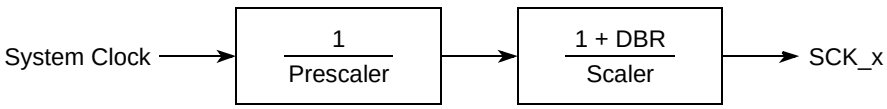


Figure 405. Communications clock prescalers and scalers

Baud rate generator

The baud rate is the frequency of the serial communication clock (SCK_*x*). The system clock is divided by a baud rate prescaler (defined by DSPI*x*_CTAR[PBR]) and baud rate scaler (defined by DSPI*x*_CTAR[BR]) to produce SCK_*x* with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI*x*_CTARs select the frequency of SCK_*x* using the following formula:

Equation 55

$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 383 shows an example of a computed baud rate.

Table 383. Baud rate computation example

f_{SYS}	PBR	Prescaler value	BR	Scaler value	DBR value	Baud rate
100 MHz	0b00	2	0b0000	2	0	25 Mbit/s
20 MHz	0b00	2	0b0000	2	1	10 Mbit/s

CS to SCK delay (t_{CSC})

The $\overline{CS}\_x$ to SCK_*x* delay is the length of time from assertion of the $\overline{CS}\_x$ signal to the first SCK_*x* edge. Refer to *Figure 407* for an illustration of the $\overline{CS}\_x$ to SCK_*x* delay. The PCSSCK and CSSCK fields in the DSPI*x*_CTAR*n* registers select the $\overline{CS}\_x$ to SCK_*x* delay, and the relationship is expressed by the following formula:

Equation 56

$$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 384 shows an example of the computed $\overline{CS}$ to SCK_*x* delay.

Table 384. $\overline{CS}$ to SCK delay computation example

PCSSCK	Prescaler value	CSSCK	Scaler value	f_{SYS}	$\overline{CS}$ to SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

After SCK delay (t_{ASC})

The after SCK_*x* delay is the length of time between the last edge of SCK_*x* and the deassertion of CS_*x*. Refer to *Figure 407* and *Figure 408* for illustrations of the after SCK_*x* delay. The PASC and ASC fields in the DSPI*x*_CTAR*n* registers select the after SCK delay. The relationship between these variables is given in the following formula:

Equation 57

$$t_{ASC} = \frac{1}{f_{SYS}} \times \text{PASC} \times \text{ASC}$$

Table 385 shows an example of the computed after SCK delay.

Table 385. After SCK delay computation example

PASC	Prescaler value	ASC	Scaler value	f_{SYS}	After SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

Delay after transfer (t_{DT})

The delay after transfer is the length of time between negation of the $\overline{CSx}$ signal for a frame and the assertion of the $\overline{CSx}$ signal for the next frame. The PDT and DT fields in the DSPI*x*_CTAR*n* registers select the delay after transfer.

Refer to *Figure 407* for an illustration of the delay after transfer.

The following formula expresses the PDT/DT/delay after transfer relationship:

Equation 58

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 386 shows an example of the computed delay after transfer.

Table 386. Delay after transfer computation example

PDT	Prescaler value	DT	Scaler value	f_{SYS}	Delay after transfer
0b01	3	0b1110	32768	100 MHz	0.98 ms

Peripheral Chip Select strobe enable ($\overline{CS5\_x}$)

The $\overline{CS5\_x}$ signal provides a delay to allow the CS*x* signals to settle after transitioning, thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI*x*_MCR, CS5_*x* provides a signal for an external demultiplexer to decode the CS4_*x* signals into as many as 32 glitch-free CS*x* signals.

Figure 406 shows the timing of the CS5_*x* signal relative to CS signals.

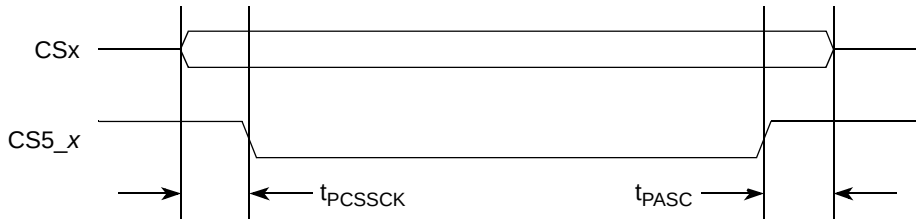


Figure 406. Peripheral Chip Select strobe timing

The delay between the assertion of the CS*x* signals and the assertion of CS5_*x* signal is selected by the PCSSCK field in the DSPI*x*_CTAR based on the following formula:

Equation 59

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between CS5_*x* negation and CS*x* negation is selected by the PASC field in the DSPI*x*_CTAR based on the following formula:

Equation 60

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 387 shows an example of the computed t_{PCSSCK} delay.

Table 387. Peripheral Chip Select strobe assert computation example

PCSSCK	Prescaler	f_{SYS}	Delay before transfer
0b11	7	100 MHz	70.0 ns

Table 388 shows an example of the computed the t_{PASC} delay.

Table 388. Peripheral Chip Select strobe negate computation example

PASC	Prescaler	f_{SYS}	Delay after transfer
0b11	7	100 MHz	70.0 ns

22.8.5 Transfer formats

The SPI serial communication is controlled by the serial communications clock (SCK_*x*) signal and the CS*x* signals. The SCK_*x* signal provided by the master device synchronizes shifting and sampling of the data by the SIN_*x* and SOUT_*x* pins. The CS*x* signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPI*x*_CTAR*n*) select the polarity and phase of the serial clock, SCK_*x*. The polarity bit selects the idle state of the SCK_*x*. The clock phase bit selects if the data on SOUT_*x* is valid before or on the first SCK_*x* edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI*x*_CTAR0 (SPI slave mode) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI*x*_MCR selects between classic SPI format and modified transfer format. The classic SPI formats are described in *Section : Classic SPI transfer format (CPHA = 0)* and *Section : Classic SPI transfer format (CPHA = 1)."* The modified transfer formats are described in *Section : Modified SPI transfer format (MTFE = 1, CPHA = 0)* and *Section : Modified SPI transfer format (MTFE = 1, CPHA = 1)."*

In the SPI configuration, the DSPI provides the option of keeping the CS signals asserted between frames. Refer to *Section : Continuous selection format* for details.

Classic SPI transfer format (CPHA = 0)

The transfer format shown in *Figure 407* is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN_*x* pins on the odd-numbered SCK_*x* edges and change the data on their SOUT_*x* pins on the even-numbered SCK_*x* edges.

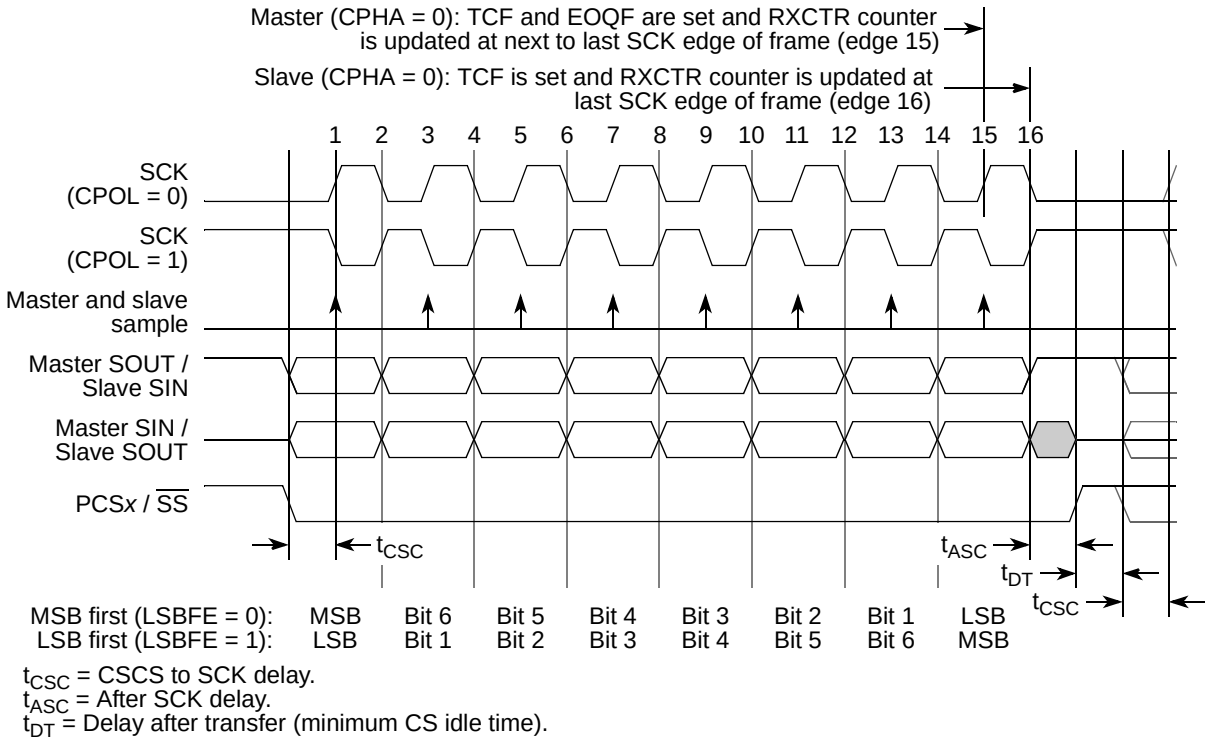


Figure 407. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

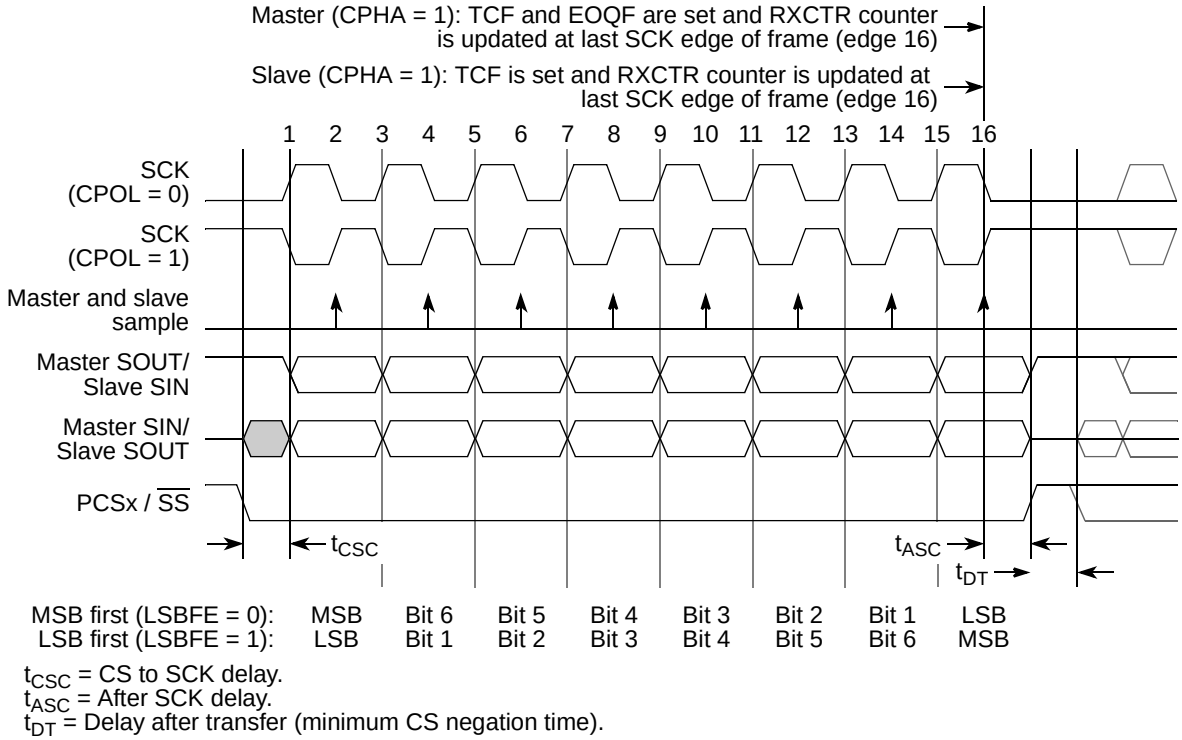
The master initiates the transfer by placing its first data bit on the SOUT_*x* pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT_*x* pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK_*x*. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK_*x* the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN_*x* pins on the odd-numbered clock edges and changes the data on their SOUT_*x* pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the CS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of *Figure 407*.

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of *Figure 407*.

Classic SPI transfer format (CPHA = 1)

The transfer format shown in *Figure 408* is used to communicate with peripheral SPI slave devices that require the first SCK_*x* edge before the first data bit becomes available on the slave SOUT_*x* pin. In this format the master and slave devices change the data on their SOUT_*x* pins on the odd-numbered SCK_*x* edges and sample the data on their SIN_*x* pins on the even-numbered SCK_*x* edges.



t_{CSC} = CS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 408. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the CS*x* signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK_*x* edge and at the same time places valid data on the master SOUT_*x* pin. The slave responds to the first SCK_*x* edge by placing its first data bit on its slave SOUT_*x* pin.

At the second edge of the SCK_*x* the master and slave sample their SIN_*x* pins. For the rest of the frame the master and the slave change the data on their SOUT_*x* pins on the odd-numbered clock edges and sample their SIN_*x* pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the CS*x* signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of *Figure 408*. For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

Modified SPI transfer format (MTFE = 1, CPHA = 0)

In this modified transfer format both the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT_*x* pins at the assertion of the CS*x* signal. After the CS*x* to SCK_*x* delay has elapsed the first SCK_*x* edge is generated. The slave samples the master SOUT_*x* signal on every odd numbered SCK_*x* edge. The slave also places new data on the slave SOUT_*x* on every odd numbered clock edge.

The master places its second data bit on the SOUT_*x* line one system clock after odd numbered SCK_*x* edge. The point where the master samples the slave SOUT_*x* is selected by writing to the SMPL_PT field in the DSPI*x*_MCR. *Table 389* lists the number of system clock cycles between the active edge of SCK_*x* and the master sample point for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 389. Delayed master sample point

SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK and sampling of SIN
00	0
01	1
10	2
11	Invalid value

Figure 409 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

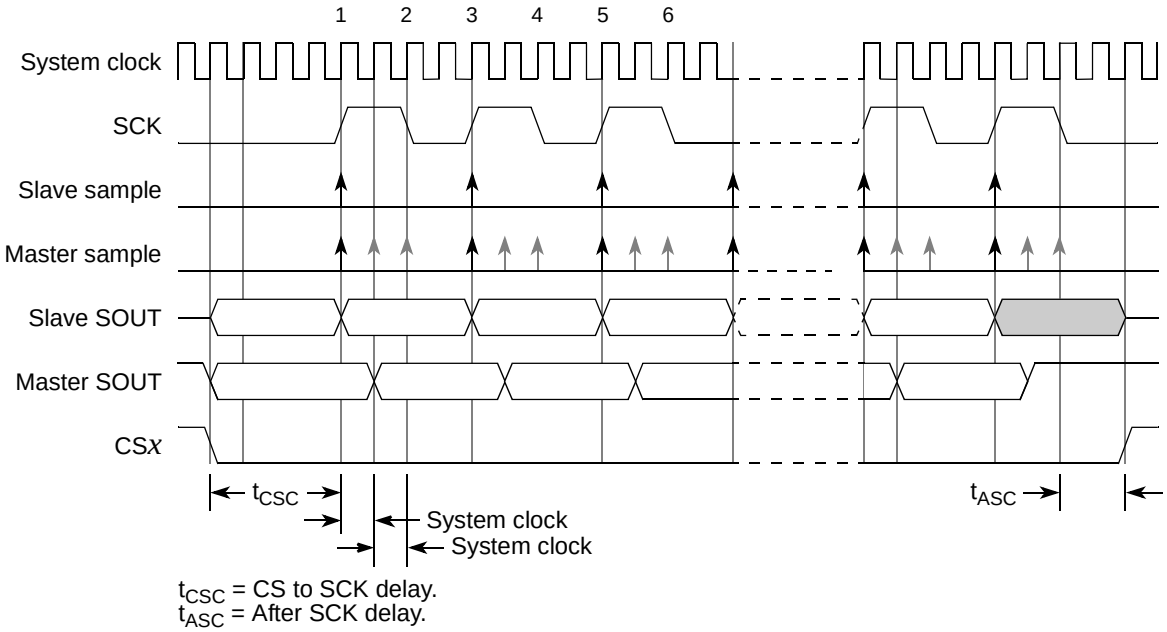


Figure 409. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$)

Modified SPI transfer format (MTFE = 1, CPHA = 1)

At the start of a transfer the DSPI asserts the CS signal to the slave device. After the CS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the 3rd SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit. The SCK to CS delay must be programmed to be greater than or equal to half the SCK period.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 410 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is shown.

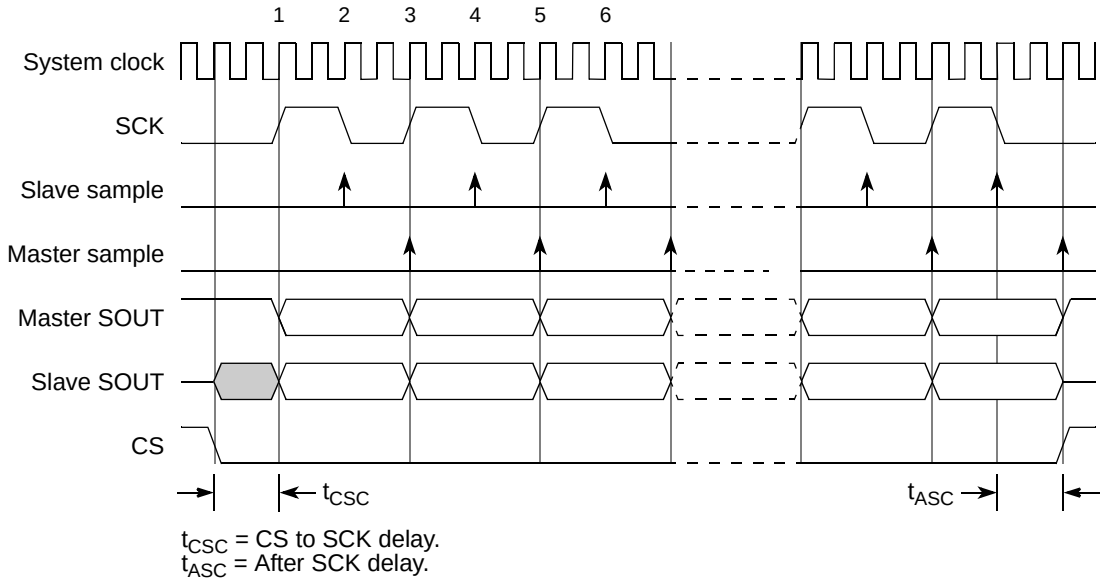


Figure 410. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$)

Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPI*x*_MCR.

Figure 411 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.

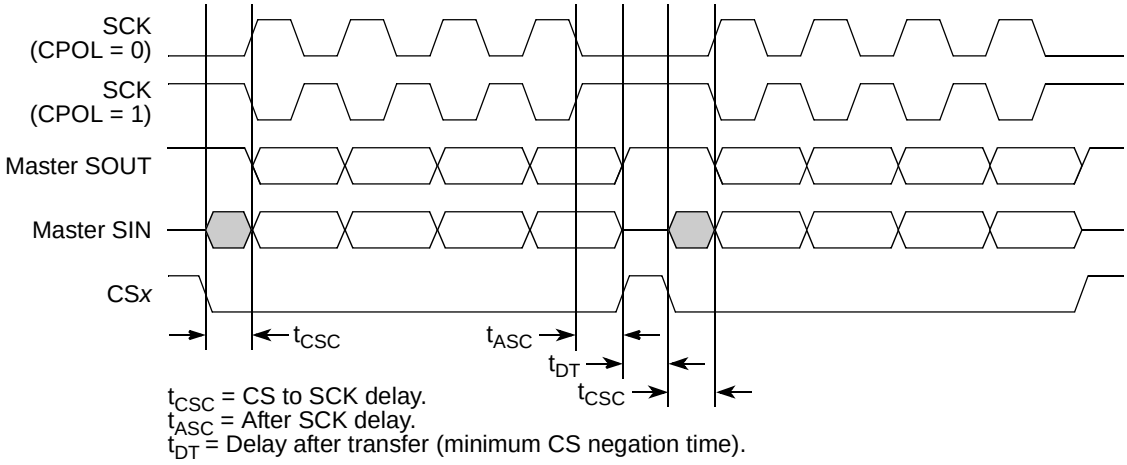


Figure 411. Example of non-continuous format (CPHA = 1, CONT = 0)

When the CONT = 1 and the CS signal for the next transfer is the same as for the current transfer, the CS signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers.

Figure 412 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

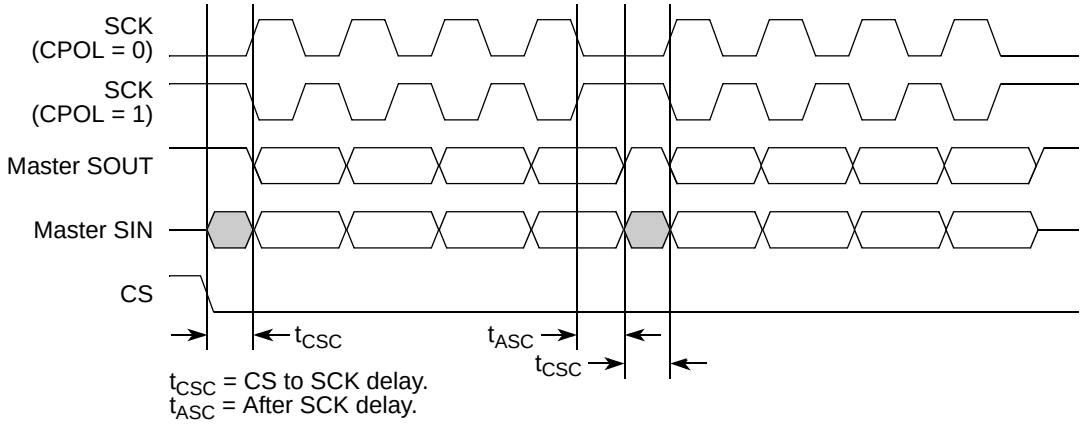


Figure 412. Example of continuous transfer (CPHA = 1, CONT = 1)

In *Figure 412*, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}; i.e., it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The CS signal must be negated before CTAR is switched.

When the CONT bit = 1 and the CS signals for the next transfer are different from the present transfer, the CS signals behave as if the CONT bit was not set.

Note: It is mandatory to fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example; while transmitting in master mode, it should be ensured that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0).While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is completely transmited (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be de-selected for any further serial communication; else an underflow error occurs

Clock polarity switching between DSPI transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame.

Refer to *Section : DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn)*.

In *Figure 413*, time ‘A’ shows the one clock interval. Time ‘B’ is user programmable from a minimum of two system clocks.

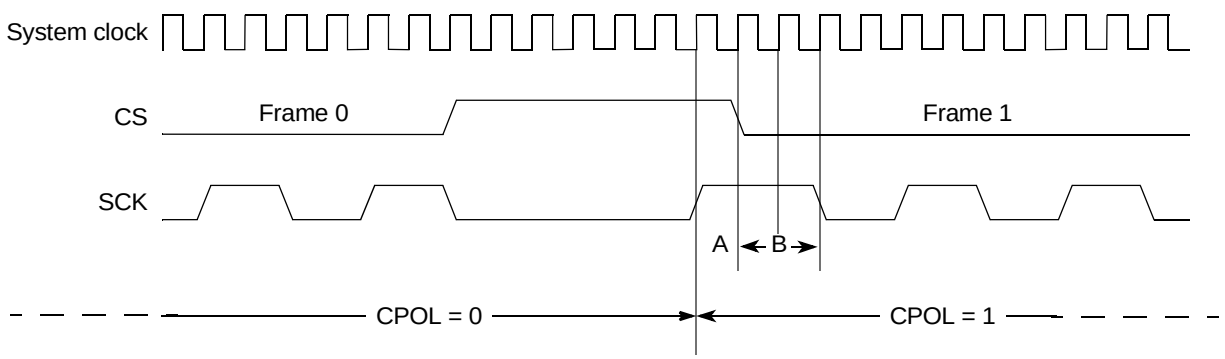


Figure 413. Polarity switching between frames

22.8.6 Continuous Serial communications clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI*x*_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame is used.
- In all configurations, the currently selected CTAR remains in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

The device is designed to use the same baud rate for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the CS to SCK delay and the After SCK delay. The delay after transfer is fixed at one SCK cycle. *Figure 414* shows timing diagram for continuous SCK format with continuous selection disabled.

Note: When in Continuous SCK mode, for the SPI transfer CTAR0 should always be used, and the TX-FIFO must be clear using the MCR.CLR_TXF field before initiating transfer.

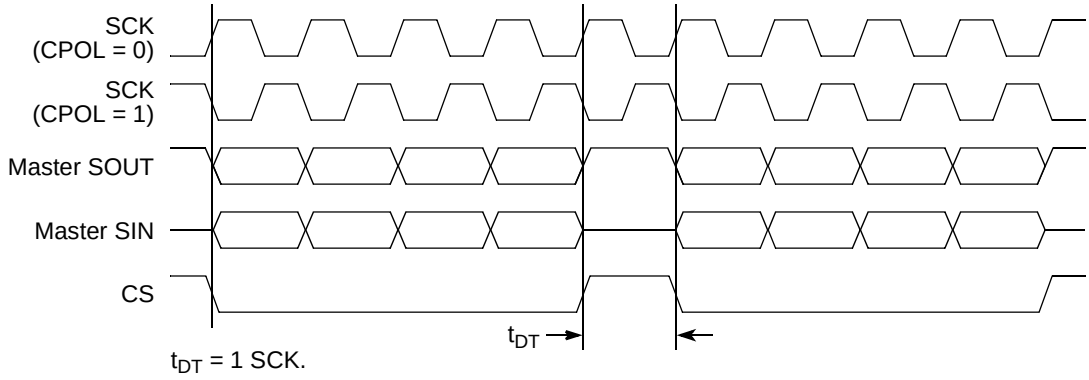


Figure 414. Continuous SCK timing diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set, CS remains asserted between the transfers when the CS signal for the next transfer is the same as for the current transfer. *Figure 415* shows timing diagram for continuous SCK format with continuous selection enabled.

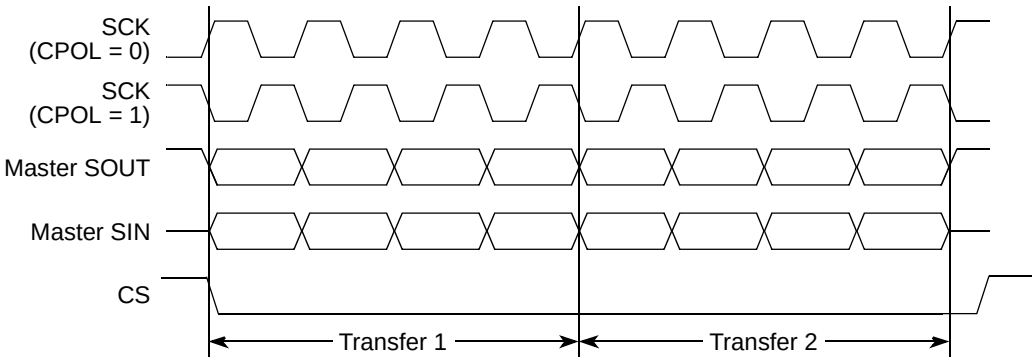


Figure 415. Continuous SCK timing diagram (CONT = 1)

22.8.7 Interrupts/DMA requests

The DSPI has conditions that can generate interrupt requests only, and conditions that can generate interrupts or DMA requests. *Table 390* lists these conditions.

Table 390. Interrupt and DMA request conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	
TX FIFO underflow has occurred	TFUF	X	
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow occurred	RFOF	X	
A FIFO overrun occurred(1)	TFUF ORed with RFOF	X	

1. The FIFO overrun condition is created by ORing the TFUF and RFOF flags together.

Each condition has a flag bit and a request enable bit. The flag bits are described in the *Section : DSPI Status Register (DSPIx_SR)* and the request enable bits are described in the *Section : DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)*. The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF_DIRS and RFDF_DIRS bits in the DSPI*x*_RSER.

End of queue interrupt request (EOQF)

The end of queue request indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF_RE bit in the DSPI*x*_RSER is set. Refer to the EOQ bit description in *Section : DSPI Status Register (DSPIx_SR)*. Refer to *Figure 407* and *Figure 408* that illustrate when EOQF is set.

Transmit FIFO fill interrupt or DMA request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF_RE bit in the DSPI*x*_RSER is set. The TFFF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

Transfer complete interrupt request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI*x*_RSER. Refer to the TCF bit description in *Section : DSPI Status Register (DSPIx_SR)*. Refer to *Figure 407* and *Figure 408*, which show when TCF is set.

Transmit FIFO underflow interrupt request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI

operating in slave mode and SPI configuration is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF_RE bit in the DSPI*x*_RSER is set, an interrupt request is generated.

Receive FIFO drain interrupt or DMA request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF_RE bit in the DSPI*x*_RSER is set. The RFDF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

Receive FIFO overflow interrupt request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF_RE bit in the DSPI*x*_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

FIFO overrun request (TFUF) or (RFOF)

The FIFO overrun request indicates that at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing together the RX FIFO overflow and TX FIFO underflow signals.

22.8.8 Power saving features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

Module disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPI*x*_MCR.In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI*x*_MCR does not have any affect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no affect. Writing to the DSPI*x*_TCR during module disable mode does not have an effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

22.9 Initialization and application information

22.9.1 Managing queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to manage queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPI*x*_SR is set.
3. The setting of the EOQF flag disables both serial transmission, and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI*x*_SR or by checking RFDF in the DSPI*x*_SR after each read operation of the DSPI*x*_POPR.
7. Modify DMA descriptor of TX and RX channels for “new” queues.
8. Flush TX FIFO by writing a 1 to the CLR_TXF bit in the DSPI*x*_MCR register and flush the RX FIFO by writing a 1 to the CLR_RXF bit in the DSPI*x*_MCR register.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI_TCNT field in the DSPI*x*_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

22.9.2 Baud rate settings

Table 391 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI*x*_CTARs. The values are calculated at a 100 MHz system frequency.

Table 391. Baud rate values

		Baud Rate divider prescaler values (DSPI_CTAR[PBR])			
		2	**3**	**5**	**7**
Baud rate scaler values (DSPI_CTAR[BR])	**2**	25.0 MHz	16.7 MHz	10.0 MHz	7.14 MHz
	4	12.5 MHz	8.33 MHz	5.00 MHz	3.57 MHz
	6	8.33 MHz	5.56 MHz	3.33 MHz	2.38 MHz
	8	6.25 MHz	4.17 MHz	2.50 MHz	1.79 MHz
	16	3.12 MHz	2.08 MHz	1.25 MHz	893 kHz
	32	1.56 MHz	1.04 MHz	625 kHz	446 kHz
	64	781 kHz	521 kHz	312 kHz	223 kHz
	128	391 kHz	260 kHz	156 kHz	112 kHz
	256	195 kHz	130 kHz	78.1 kHz	55.8 kHz
	512	97.7 kHz	65.1 kHz	39.1 kHz	27.9 kHz
	1024	48.8 kHz	32.6 kHz	19.5 kHz	14.0 kHz
	2048	24.4 kHz	16.3 kHz	9.77 kHz	6.98 kHz
	4096	12.2 kHz	8.14 kHz	4.88 kHz	3.49 kHz
	8192	6.10 kHz	4.07 kHz	2.44 kHz	1.74 kHz
	16384	3.05 kHz	2.04 kHz	1.22 kHz	872 Hz
	32768	1.53 kHz	1.02 kHz	610 Hz	436 Hz

22.9.3 Delay settings

Table 392 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI*x*_CTARs. The values calculated assume a 100 MHz system frequency.

Table 392. Delay values

		Delay prescaler values (DSPI_CTAR[PBR])			
		1	**3**	**5**	**7**
Delay scaler values (DSPI_CTAR[DT])	**2**	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

22.9.4 Calculation of FIFO pointer addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR).

Refer to *Section : Transmit First In First Out (TX FIFO) buffering mechanism* and *Section : Receive First In First Out (RX FIFO) buffering mechanism* for details on the FIFO operation. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO.

Figure 416 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

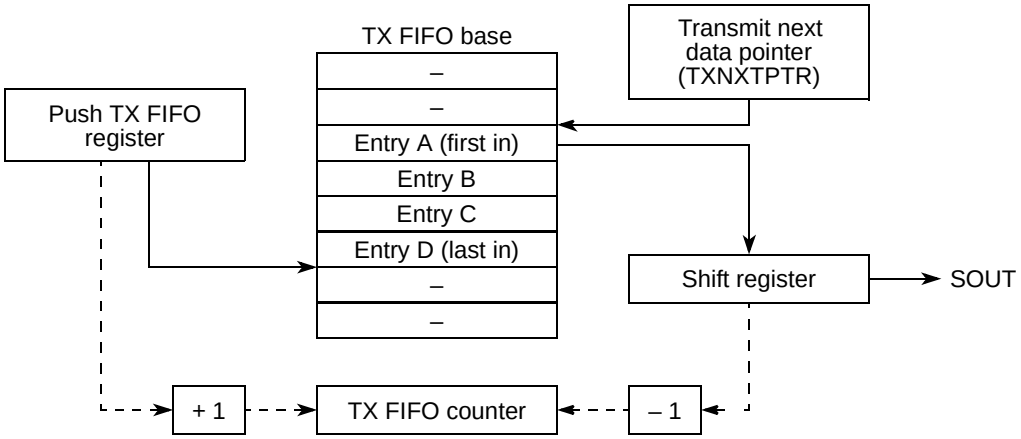


Figure 416. TX FIFO pointers and counter

Address calculation for first-in entry and last-in entry in TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

Equation 61

$$\text{First-in entry address} = \text{TXFIFO base} + 4\ (\text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 62

$$\text{Last-in entry address} = \text{TXFIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXTPTR} - 1) \text{ modulo TXFIFO depth}]$$

where:

TXFIFO base = base address of transmit FIFO

TXCTR = transmit FIFO counter

TXNXTPTR = transmit next pointer

TX FIFO depth = transmit FIFO depth (depth is 5)

Address calculation for first-in entry and last-in entry in RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

Equation 63

$$\text{First-in entry address} = \text{RXFIFO base} + 4 \times (\text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 64

$$\text{Last-in entry address} = \text{RXFIFO base} + 4 \times [(\text{RXCTR} + \text{POPNXTPTR} - 1) \text{ modulo RXFIFO depth}]$$

where:

RXFIFO base = base address of receive FIFO

RXCTR = receive FIFO counter

POPNXTPTR = pop next pointer

RX FIFO depth = receive FIFO depth (depth is 5)

23 LIN Controller (LINFlex)

23.1 Introduction

The LINFlex (Local Interconnect Network Flexible) controller interfaces the LIN network and supports the LIN protocol versions 1.3 and 2.0 in both Master and Slave modes. LINFlex includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load and allow slave node resynchronization.

23.2 Main features

23.2.1 LIN mode features

- Supports LIN protocol versions 1.3 and 2.0
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for In-Application Programming (IAP) purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with 16 MHz fast internal RC oscillator as clock source
- 16 identifier filters for autonomous message handling in Slave mode

23.2.2 UART mode features

- Full duplex communication
- 8- or 9-bit with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

23.2.3 Features common to LIN and UART

- Fractional baud rate generator
- 3 operating modes for power saving and configuration registers lock:
  - Initialization
  - Normal
  - Sleep
- 2 test modes:
  - Loop Back
  - Self Test
- Maskable interrupts

23.3 General description

The increasing number of communication peripherals embedded on microcontrollers, for example CAN, LIN and SPI, requires more and more CPU resources for communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high-level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 Kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as usually the case.

To minimize the CPU load in Master mode, LINFlex handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlex does not request any software intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlex requires software intervention only to:

- Trigger transmission or reception or data discard depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for Slave mode, LINFlex requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode)

The software uses the control, status and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

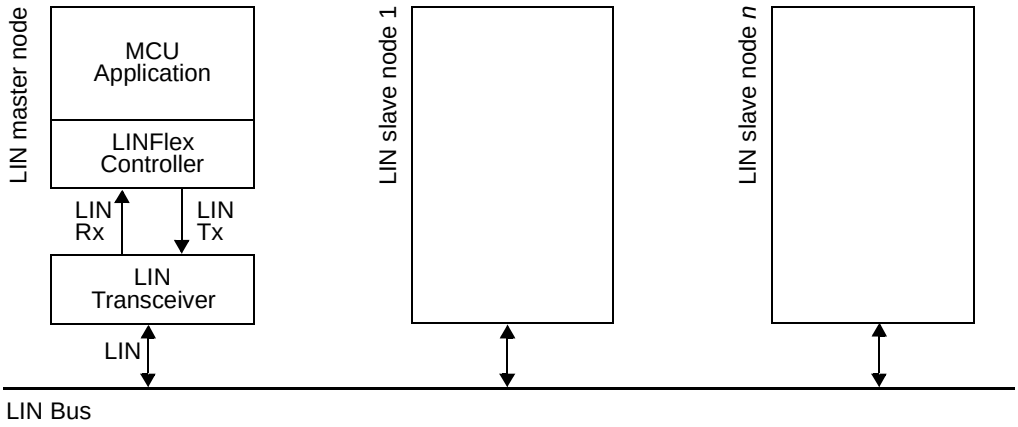
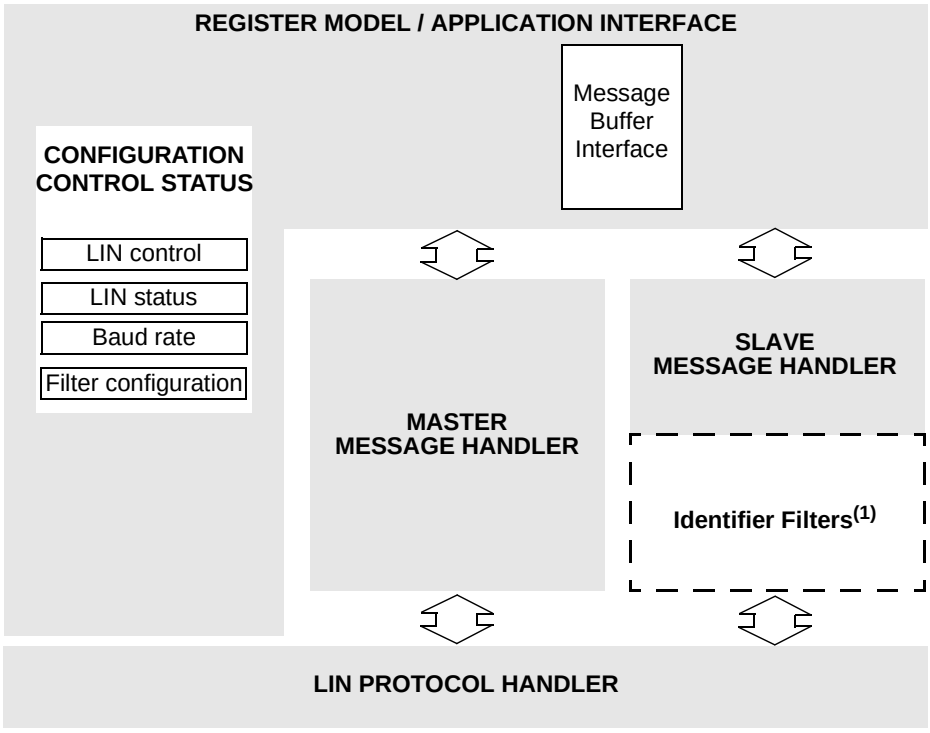


Figure 417. LIN topology network



1. Filter activation optional

Figure 418. LINFlex block diagram

23.4 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBRR) registers.

Equation 65

$$\text{Tx/ Rx baud} = \frac{f_{periph\_set\_1\_clk}}{(16 \times LFDIV)}$$

LFDIV is an unsigned fixed point number. The 12-bit mantissa is coded in the LINIBRR and the fraction is coded in the LINFBRR.

The following examples show how to derive LFDIV from LINIBRR and LINFBRR register values:

Example 11 Deriving LFDIV from LINIBRR and LINFBRR register values

If LINIBRR = 27d and LINFBRR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

Example 12 Programming LFDIV from LINIBRR and LINFBRR register values

To program LFDIV = 25.62d,

LINFBRR = 16 × 0.62 = 9.92, nearest real number 10d = 0xA

LINIBRR = mantissa (25.620d) = 25d = 0x19

Note: The baud counters are updated with the new value of the baud registers after a write to LINIBRR. Hence the baud register value must not be changed during a transaction. The LINFBRR (containing the Fraction bits) must be programmed before the LINIBRR.

Note: LFDIV must be greater than or equal to 1.5d, i.e. LINIBRR = 1 and LINFBRR = 8. Therefore, the maximum possible baudrate is fperiph_set_1_clk / 24.

Table 393. Error calculation for programmed baud rates

Baud rate	$f_{periph\_set\_1\_clk}$ = 64 MHz				$f_{periph\_set\_1\_clk}$ = 16 MHz			
	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate
		LINIBRR	LINFBRR			LINIBRR	LINFBRR	
2400	2399.97	1666	11	–0.001	2399.88	416	11	–0.005
9600	9599.52	416	11	–0.005	9598.08	104	3	–0.02
10417	10416.7	384	0	–0.003	10416.7	96	0	–0.003
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	–0.08
115200	115108	34	12	–0.08	115108	8	11	–0.08
230400	230216	17	6	–0.08	231884	4	5	0.644
460800	460432	8	11	–0.08	457143	2	3	–0.794
921600	927536	4	5	0.644	941176	1	1	2.124

23.5 Operating modes

LINFlex has three main operating modes: Initialization, Normal and Sleep. After a hardware reset, LINFlex is in Sleep mode to reduce power consumption. The software instructs LINFlex to enter Initialization mode or Sleep mode by setting the INIT bit or SLEEP bit in the LINCR1.

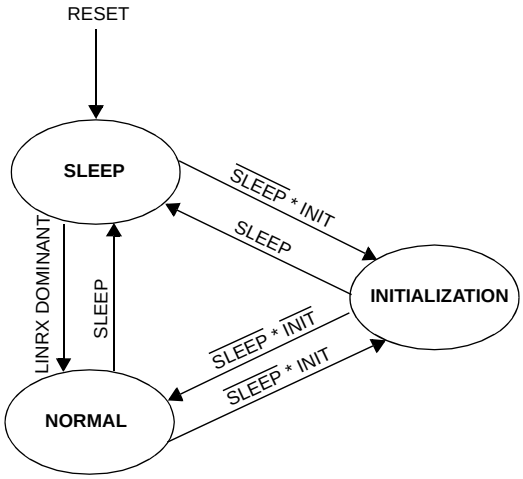


Figure 419. LINFlex operating modes

23.5.1 Initialization mode

The software can be initialized while the hardware is in Initialization mode. To enter this mode the software sets the INIT bit in the LINCR1.

To exit Initialization mode, the software clears the INIT bit.

While in Initialization mode, all message transfers to and from the LIN bus are stopped and the status of the LIN bus output LINTX is recessive (high).

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlex controller, the software selects the mode (LIN Master, LIN Slave or UART), sets up the baud rate register and, if LIN Slave mode with filter activation is selected, initializes the identifier list.

23.5.2 Normal mode

Once initilization is complete, software clears the INIT bit in the LINCR1 to put the hardware into Normal mode.

23.5.3 Low power mode (Sleep)

To reduce power consumption, LINFlex has a low power mode called Sleep mode. To enter Sleep mode, software sets the SLEEP bit in the LINCR1. In this mode, the LINFlex clock is stopped. Consequently, the LINFlex will not update the status bits but software can still access the LINFlex registers.

LINFlex can be awakened (exit Sleep mode) either by software clearing the SLEEP bit or on detection of LIN bus activity if automatic wake-up mode is enabled (AWUM bit is set).

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the LINCR1 is set. To exit from Sleep mode if the AWUM bit is cleared, software clears the SLEEP bit when a wake-up event occurs.

23.6 Test modes

Two test modes are available to the user: Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1. These bits must be configured while LINFlex is in Initialization mode. Once one of the two test modes has been selected, LINFlex must be started in Normal mode.

23.6.1 Loop Back mode

LINFlex can be put in Loop Back mode by setting the LBKM bit in the LINCR. In Loop Back mode, the LINFlex treats its own transmitted messages as received messages.

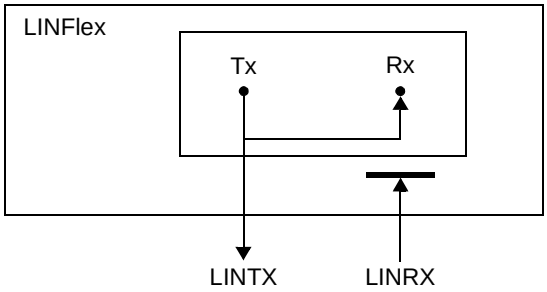


Figure 420. LINFlex in loop back mode

This mode is provided for self test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlex performs an internal feedback from its Tx output to its Rx input. The actual value of the LINRX input pin is disregarded by the LINFlex. The transmitted messages can be monitored on the LINTX pin.

23.6.2 Self Test mode

LINFlex can be put in Self Test mode by setting the LBKM and SFTM bits in the LINCR. This mode can be used for a “Hot Self Test”, meaning the LINFlex can be tested as in Loop Back mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlex and the LINTX pin is held recessive.

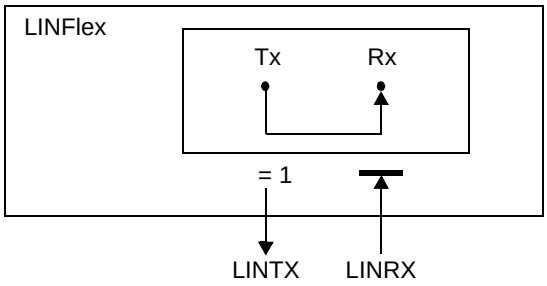


Figure 421. LINFlex in self test mode

23.7 Memory map and registers description

23.7.1 Memory map

See the “Memory map” chapter of this reference manual for the base addresses for the LINFlex modules.

Table 394 shows the LINFlex memory map.

Table 394. LINFlex memory map

Address offset	Register	Location
0x0000	LIN control register 1 (LINCR1)	*on page 727*
0x0004	LIN interrupt enable register (LINIER)	*on page 730*
0x0008	LIN status register (LINSR)	*on page 732*
0x000C	LIN error status register (LINESR)	*on page 735*
0x0010	UART mode control register (UARTCR)	*on page 736*
0x0014	UART mode status register (UARTSR)	*on page 738*
0x0018	LIN timeout control status register (LINTCSR)	*on page 740*
0x001C	LIN output compare register (LINOCR)	*on page 741*
0x0020	LIN timeout control register (LINTOCR)	*on page 741*
0x0024	LIN fractional baud rate register (LINFBRR)	*on page 742*
0x0028	LIN integer baud rate register (LINIBRR)	*on page 743*
0x002C	LIN checksum field register (LINCFR)	*on page 744*
0x0030	LIN control register 2 (LINCR2)	*on page 744*
0x0034	Buffer identifier register (BIDR)	*on page 746*
0x0038	Buffer data register LSB (BDRL)(1)	*on page 747*
0x003C	Buffer data register MSB (BDRM)(2)	*on page 747*
0x0040	Identifier filter enable register (IFER)	*on page 748*

Table 394. LINFlex memory map (continued)

Address offset	Register	Location
0x0044	Identifier filter match index (IFMI)	*on page 749*
0x0048	Identifier filter mode register (IFMR)	*on page 749*
0x004C	Identifier filter control register 0 (IFCR0)	*on page 751*
0x0050	Identifier filter control register 1 (IFCR1)	*on page 752*
0x0054	Identifier filter control register 2 (IFCR2)	*on page 752*
0x0058	Identifier filter control register 3 (IFCR3)	*on page 752*
0x005C	Identifier filter control register 4 (IFCR4)	*on page 752*
0x0060	Identifier filter control register 5 (IFCR5)	*on page 752*
0x0064	Identifier filter control register 6 (IFCR6)	*on page 752*
0x0068	Identifier filter control register 7 (IFCR7)	*on page 752*
0x006C	Identifier filter control register 8 (IFCR8)	*on page 752*
0x0070	Identifier filter control register 9 (IFCR9)	*on page 752*
0x0074	Identifier filter control register 10 (IFCR10)	*on page 752*
0x0078	Identifier filter control register 11 (IFCR11)	*on page 752*
0x007C	Identifier filter control register 12 (IFCR12)	*on page 752*
0x0080	Identifier filter control register 13 (IFCR13)	*on page 752*
0x0084	Identifier filter control register 14 (IFCR14)	*on page 752*
0x0088	Identifier filter control register 15 (IFCR15)	*on page 752*
0x008C–0x3FFF	Reserved	

1. LSB: Least significant byte
2. MSB: Most significant byte

LIN control register 1 (LINCR1)

Figure 422. LIN control register 1 (LINCR1)

Offset: 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CCD	CFD	LASE	AWUM	MBL				BF	SFTM	LBKM	MME	SBDT	RBLM	SLEEP	INIT
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Table 395. LINCR1 field descriptions

Field	Description
CCD	Checksum calculation disable This bit disables the checksum calculation (see *Table 396*). 0 Checksum calculation is done by hardware. When this bit is 0, the LINCFR is read-only. 1 Checksum calculation is disabled. When this bit is set the LINCFR is read/write. User can program this register to send a software-calculated CRC (provided CFD is 0). Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
CFD	Checksum field disable This bit disables the checksum field transmission (see *Table 396*). 0 Checksum field is sent after the required number of data bytes is sent. 1 No checksum field is sent. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
LASE	LIN Slave Automatic Resynchronization Enable 0 Automatic resynchronization disable. 1 Automatic resynchronization enable. This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
AWUM	Automatic Wake-Up Mode This bit controls the behavior of the LINFlex hardware during Sleep mode. 0 The Sleep mode is exited on software request by clearing the SLEEP bit of the LINCR. 1 The Sleep mode is exited automatically by hardware on LINRX dominant state detection. The SLEEP bit of the LINCR is cleared by hardware whenever WUF bit in the LINSR is set. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
MBL	LIN Master Break Length This field indicates the Break length in Master mode (see *Table 397*). Note: This field can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
BF	Bypass filter 0 No interrupt if identifier does not match any filter. 1 An RX interrupt is generated on identifier not matching any filter. – If no filter is activated, this bit is reserved. – This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SFTM	Self Test Mode This bit controls the Self Test mode. For more details, see *Section 23.6.2, Self Test mode.* 0 Self Test mode disable. 1 Self Test mode enable. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.

Table 395. LINCR1 field descriptions (continued)

Field	Description
LBKM	Loop Back Mode This bit controls the Loop Back mode. For more details see *Section 23.6.1, Loop Back mode.* 0 Loop Back mode disable. 1 Loop Back mode enable. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode
MME	Master Mode Enable 0 Slave mode enable. 1 Master mode enable. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SBDT	Slave Mode Break Detection Threshold 0 11-bit break. 1 10-bit break. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
RBLM	Receive Buffer Locked Mode 0 Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one. 1 Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SLEEP	Sleep Mode Request This bit is set by software to request LINFlex to enter Sleep mode. This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see *Table 398*).
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlex enters Normal mode when clearing the INIT bit (see *Table 398*).

Table 396. Checksum bits configuration

CFD	CCD	LINCFR	Checksum sent
1	1	Read/Write	None
1	0	Read-only	None
0	1	Read/Write	Programmed in LINCFR by bits CF[0:7]
0	0	Read-only	Hardware calculated

Table 397. LIN master break length selection

MBL	Length
0000	10-bit

Table 397. LIN master break length selection (continued)

MBL	Length
0001	11-bit
0010	12-bit
0011	13-bit
0100	14-bit
0101	15-bit
0110	16-bit
0111	17-bit
1000	18-bit
1001	19-bit
1010	20-bit
1011	21-bit
1100	22-bit
1101	23-bit
1110	36-bit
1111	50-bit

Table 398. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

LIN interrupt enable register (LINIER)

Figure 423. LIN interrupt enable register (LINIER)

Offset: 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZIE	OCIE	BEIE	CEIE	HEIE	0	0	FEIE	BOIE	LSIE	WUIE	DBFIE	DBEIE	DRIE	DTIE	HRIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 399. LINIER field descriptions

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0 No interrupt when SZF bit in LINESR or UARTSR is set. 1 Interrupt generated when SZF bit in LINESR or UARTSR is set.
OCIE	Output Compare Interrupt Enable 0 No interrupt when OCF bit in LINESR or UARTSR is set. 1 Interrupt generated when OCF bit in LINESR or UARTSR is set.
BEIE	Bit Error Interrupt Enable 0 No interrupt when BEF bit in LINESR is set. 1 Interrupt generated when BEF bit in LINESR is set.
CEIE	Checksum Error Interrupt Enable 0 No interrupt on Checksum error. 1 Interrupt generated when checksum error flag (CEF) in LINESR is set.
HEIE	Header Error Interrupt Enable 0 No interrupt on Break Delimiter error, Synch Field error, Identifier field error. 1 Interrupt generated on Break Delimiter error, Synch Field error, Identifier field error.
FEIE	Framing Error Interrupt Enable 0 No interrupt on Framing error. 1 Interrupt generated on Framing error.
BOIE	Buffer Overrun Interrupt Enable 0 No interrupt on Buffer overrun. 1 Interrupt generated on Buffer overrun.
LSIE	LIN State Interrupt Enable 0 No interrupt on LIN state change. 1 Interrupt generated on LIN state change. This interrupt can be used for debugging purposes. It has no status flag but is reset when writing ‘1111’ into LINS[0:3] in the LINSR.
WUIE	Wake-up Interrupt Enable 0 No interrupt when WUF bit in LINSR or UARTSR is set. 1 Interrupt generated when WUF bit in LINSR or UARTSR is set.
DBFIE	Data Buffer Full Interrupt Enable 0 No interrupt when buffer data register is full. 1 Interrupt generated when data buffer register is full.
DBEIE	Data Buffer Empty Interrupt Enable 0 No interrupt when buffer data register is empty. 1 Interrupt generated when data buffer register is empty.
DRIE	Data Reception Complete Interrupt Enable 0 No interrupt when data reception is completed. 1 Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set.
DTIE	Data Transmitted Interrupt Enable 0 No interrupt when data transmission is completed. 1 Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR.

Table 399. LINIER field descriptions (continued)

Field	Description
HRIE	Header Received Interrupt Enable 0 No interrupt when a valid LIN header has been received. 1 Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR is set.

LIN status register (LINSR)

Figure 424. LIN status register (LINSR)

Offset: 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LINS				0	0	RMB	0	RBSY	RPS	WUF	DBFF	DBEF	DRF	DTF	HRF
W							w1c		w1c		w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Table 400. LINSR field descriptions

Field	Description
LINS	LIN modes / normal mode states **0000: Sleep mode** LINFlex is in Sleep mode to save power consumption. **0001: Initialization mode** LINFlex is in Initialization mode. Normal mode states **0010: Idle** This state is entered on several events: – SLEEP bit and INIT bit in LINCR1 have been cleared by software, – A falling edge has been received on RX pin and AWUM bit is set, – The previous frame reception or transmission has been completed or aborted. **0011: Break** In Slave mode, a falling edge followed by a dominant state has been detected. Receiving Break. Note: In Slave mode, in case of error new LIN state can be either Idle or Break depending on last bit state. If last bit is dominant new LIN state is Break, otherwise Idle.In Master mode, Break transmission ongoing. **0100: Break Delimiter** In Slave mode, a valid Break has been detected. See *Section , LIN control register 1 (LINCR1)* for break length configuration (10-bit or 11-bit). Waiting for a rising edge. In Master mode, Break transmission has been completed. Break Delimiter transmission is ongoing. **0101: Synch Field** In Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field. In Master mode, Synch Field transmission is ongoing. **0110: Identifier Field** In Slave mode, a valid Synch Field has been received. Receiving Identifier Field. In Master mode, identifier transmission is ongoing. **0111: Header reception/transmission completed** In Slave mode, a valid header has been received and identifier field is available in the BIDR. In Master mode, header transmission is completed. **1000: Data reception/transmission** Response reception/transmission is ongoing. **1001: Checksum** Data reception/transmission completed. Checksum reception/transmission ongoing. In UART mode, only the following states are flagged by the LIN state bits: – Init – Sleep – Idle – Data transmission/reception

Table 400. LINSR field descriptions (continued)

Field	Description
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.
RBSY	Receiver Busy Flag 0 Receiver is idle 1 Reception ongoing Note: In Slave mode, after header reception, if BIDR[DIR] = 0 and reception starts then this bit is set. In this case, user cannot program LINCR2[DTRQ] = 1.
RPS	LIN receive pin state This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin when: – Slave is in Sleep mode – Master is in Sleep mode or idle state This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.
DBFF	Data Buffer Full Flag This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7). This bit must be cleared by software. It is reset by hardware in Initialization mode.
DBEF	Data Buffer Empty Flag This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7). This bit must be cleared by software, once buffer has been filled again, in order to start transmission. This bit is reset by hardware in Initialization mode.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. Note: This flag is not set in case of bit error or framing error.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. Note: This flag is not set in case of bit error if IOBE bit is reset.

Table 400. LINSR field descriptions (continued)

Field	Description
HRF	Header Reception Flag This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software. This bit is reset by hardware in Initialization mode and at end of completed or aborted frame. Note: If filters are enabled, this bit is set only when identifier software filtering is required, that is to say:–All filters are inactive and BF bit in LINCR1 is set – No match in any filter and BF bit in LINCR1 is set – TX filter match

LIN error status register (LINESR)

Figure 425. LIN error status register (LINESR)

Offset: 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	BEF	CEF	SFEF	BDEF	IDPEF	FEF	BOF	0	0	0	0	0	0	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 401. LINESR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. If the dominant state continues, SZF flag is set again after 87-bit time. It is cleared by software.
OCF	Output Compare Flag 0 No output compare event occurred 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. If this bit is set and IOT bit in LINTCSR is set, LINFlex moves to Idle state. If LTOM bit in LINTCSR is set, then OCF is cleared by hardware in Initialization mode. If LTOM bit is cleared, then OCF maintains its status whatever the mode is.
BEF	Bit Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a bit error. This error can occur during response field transmission (Slave and Master modes) or during header transmission (in Master mode). This bit is cleared by software.

Table 401. LINESR field descriptions (continued)

Field	Description
CEF	Checksum Error Flag This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum. This bit is cleared by software. Note: This bit is never set if CCD or CFD bit in LINCR1 is set.
SFEF	Synch Field Error Flag This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch Field).
BDEF	Break Delimiter Error Flag This bit is set by hardware and indicates that the received Break Delimiter is too short (less than one bit time).
IDPEF	Identifier Parity Error Flag This bit is set by hardware and indicates that a Identifier Parity error occurred. Note: Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (Master or Slave mode) or during reception of Synch Field or Identifier Field in Slave mode.
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It can be cleared by software.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

UART mode control register (UARTCR)

Figure 426. UART mode control register (UARTCR)

Offset: 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	TDFL		0	RDFL		0	0	0	0	RXEN	TXEN	OP	PCE	WL	UART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 402. UARTCR field descriptions

Field	Description
TDFL	Transmitter Data Field length This field sets the number of bytes to be transmitted in UART mode. It can be programmed only when the UART bit is set. TDFL[0:1] = Transmit buffer size – 1. 00 Transmit buffer size = 1. 01 Transmit buffer size = 2. 10 Transmit buffer size = 3. 11 Transmit buffer size = 4.
RDFL	Receiver Data Field length This field sets the number of bytes to be received in UART mode. It can be programmed only when the UART bit is set. RDFL[0:1] = Receive buffer size – 1. 00 Receive buffer size = 1. 01 Receive buffer size = 2. 10 Receive buffer size = 3. 11 Receive buffer size = 4.
RXEN	Receiver Enable 0 Receiver disable. 1 Receiver enable. This bit can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0 Transmitter disable. 1 Transmitter enable. This bit can be programmed only when the UART bit is set. Note: Transmission starts when this bit is set and when writing DATA0 in the BDRL register.
OP	Odd Parity 0 Sent parity is even. 1 Sent parity is odd. This bit can be programmed in Initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0 Parity transmit/check disable. 1 Parity transmit/check enable. This bit can be programmed in Initialization mode only when the UART bit is set.
WL	Word Length in UART mode 0 7-bit data + parity bit. 1 8-bit data (or 9-bit if PCE is set). This bit can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0 LIN mode. 1 UART mode. This bit can be programmed in Initialization mode only.

UART mode status register (UARTSR)

Figure 427. UART mode status register (UARTSR)

Offset: 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	0	DRF	DTF	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 403. UARTSR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	OCF Output Compare Flag 0 No output compare event occurred. 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. An interrupt is generated if the OCIE bit in LINIER register is set.
PE3	Parity Error Flag Rx3 This bit indicates if there is a parity error in the corresponding received byte (Rx3). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE2	Parity Error Flag Rx2 This bit indicates if there is a parity error in the corresponding received byte (Rx2). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE1	Parity Error Flag Rx1 This bit indicates if there is a parity error in the corresponding received byte (Rx1). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE0	Parity Error Flag Rx0 This bit indicates if there is a parity error in the corresponding received byte (Rx0). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.

Table 403. UARTSR field descriptions (continued)

Field	Description
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit).
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites buffer. it can be cleared by software.
RPS	LIN Receive Pin State This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin in Sleep mode. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt i generated if WUIE bit in LINIER is set.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed, that is, the number of bytes programmed in RDFL[0:1] in UARTCR have been received. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DRIE bit in LINIER is set. In UART mode, this flag is set in case of framing error, parity error or overrun.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed, that is, the number of bytes programmed in TDFL[0:1] have been transmitted. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DTIE bit in LINIER is set.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

LIN timeout control status register (LINTCSR)

Figure 428. LIN timeout control status register (LINTCSR)

Offset: 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	LTOM	IOT	TOCE	CNT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Table 404. LINTCSR field descriptions

Field	Description
LTOM	LIN timeout mode 0 LIN timeout mode (header, response and frame timeout detection). 1 Output compare mode. This bit can be set/cleared in Initialization mode only.
IOT	Idle on Timeout 0 LIN state machine not reset to Idle on timeout event. 1 LIN state machine reset to Idle on timeout event. This bit can be set/cleared in Initialization mode only.
TOCE	Timeout counter enable 0 Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event. 1 Timeout counter enable. OCF bit is set if an output compare event occurs. TOCE bit is configurable by software in Initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit.
CNT	Counter Value This field indicates the LIN timeout counter value.

LIN output compare register (LINOCR)

Figure 429. LIN output compare register (LINOCR)

Offset: 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OC2[1]								OC1[1]							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1. If LINTCSR[LTOM] = 0, this field is read-only.

Table 405. LINOCR field descriptions

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.
OC1	Output compare 1 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.

LIN timeout control register (LINTOCR)

Figure 430. LIN timeout control register (LINTOCR)

Offset: 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	RTO				0	HTO						
W																
Reset	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	0

Table 406. LINTOCR field descriptions

Field	Description
RTO	Response timeout value This field contains the response timeout duration (in bit time) for 1 byte. The reset value is 0xE = 14, corresponding to $T_{Response\_Maximum} = 1.4 \times T_{Response\_Nominal}$
HTO	Header timeout value This field contains the header timeout duration (in bit time). This value does not include the Break and the Break Delimiter. The reset value is the 0x2C = 44, corresponding to $T_{Header\_Maximum.}$ Programming LINSR[MME] = 1 changes the HTO value to 0x1C = 28. This field can be written only in Slave mode.

LIN fractional baud rate register (LINFBRR)

Figure 431. LIN fractional baud rate register (LINFBRR)

Offset: 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIV_F			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 407. LINFBRR field descriptions

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlex divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16. This field can be written in Initialization mode only.

LIN integer baud rate register (LINIBRR)

Figure 432. LIN integer baud rate register (LINIBRR)

Offset: 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	DIV_M												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 408. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa This field defines the LINFlex divider (LFDIV) mantissa value (see *Table 409*). This field can be written in Initialization mode only.

Table 409. Integer baud rate selection

DIV_M[0:12]	Mantissa
0x0000	LIN clock disabled
0x0001	1
...	...
0x1FFE	8190
ox1FFF	8191

LIN checksum field register (LINCFR)

Figure 433. LIN checksum field register (LINCFR)

Offset: 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 410. LINCFR field descriptions

Field	Description
CF	Checksum bits When LINCR1[CCD] = 0, this field is read-only. When LINCR1[CCD] = 1, this field is read/write. See *Table 396*.

LIN control register 2 (LINCR2)

Figure 434. LIN control register 2 (LINCR2)

Offset: 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	IOBE	IOPE	0	0	0	0	0	0	0	0	0	0	0	0	0
W				WURQ	DDRQ	DTRQ	ABRQ	HTRQ								
Reset	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 411. LINCR2 field descriptions

Field	Description
IOBE	Idle on Bit Error 0 Bit error does not reset LIN state machine. 1 Bit error reset LIN state machine. This bit can be set/cleared in Initialization mode only.
IOPE	Idle on Identifier Parity Error 0 Identifier Parity error does not reset LIN state machine. 1 Identifier Parity error reset LIN state machine. This bit can be set/cleared in Initialization mode only.
WURQ	Wake-up Generation Request Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in Sleep mode. Software has to exit Sleep mode before requesting a wake-up. Bit error is not checked when transmitting the wake-up request.
DDRQ	Data Discard Request Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFlex has moved to idle state. In Slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.
DTRQ	Data Transmission Request Set by software in Slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set. Cleared by hardware when the request has been completed or aborted or on an error condition. In Master mode, this bit is set by hardware when BIDR[DIR] = 1 and header transmission is completed.
ABRQ	Abort Request Set by software to abort the current transmission. Cleared by hardware when the transmission has been aborted. LINFlex aborts the transmission at the end of the current bit. This bit can also abort a wake-up request. It can also be used in UART mode.
HTRQ	Header Transmission Request Set by software to request the transmission of the LIN header. Cleared by hardware when the request has been completed or aborted. This bit has no effect in UART mode.

Buffer identifier register (BIDR)

Figure 435. Buffer identifier register (BIDR)

Offset: 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DFL						DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 412. BIDR field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1. Normally, LIN uses only DFL[2:0] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[2:0] only. DFL[5:3] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDR registers. 1 LINFlex transmits the data from the BDR registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier. In LIN slave mode (MME bit cleared in LINCR1), this bit must be configured before the header reception. If the slave has to manage frames with 2 types of checksum, filters must be configured.
ID	Identifier Identifier part of the identifier field without the identifier parity.

Buffer data register LSB (BDRL)

Figure 436. Buffer data register LSB (BDRL)

Offset: 0x0038 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA3								DATA2							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA1								DATA0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 413. BDRL field descriptions

Field	Description
DATA3	Data Byte 3 Data byte 3 of the data field.
DATA2	Data Byte 2 Data byte 2 of the data field.
DATA1	Data Byte 1 Data byte 1 of the data field.
DATA0	Data Byte 0 Data byte 0 of the data field.

Buffer data register MSB (BDRM)

Figure 437. Buffer data register MSB (BDRM)

Offset: 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA7								DATA6							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA5								DATA4							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 414. BDRM field descriptions

Field	Description
DATA7	Data Byte 7 Data byte 7 of the data field.
DATA6	Data Byte 6 Data byte 6 of the data field.
DATA5	Data Byte 5 Data byte 5 of the data field.
DATA4	Data Byte 4 Data byte 4 of the data field.

Identifier filter enable register (IFER)

Figure 438. Identifier filter enable register (IFER)

Offset: 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FACT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 415. IFER field descriptions

Field	Description
FACT	Filter activation The software sets the bit FACT[x] to activate the filters x in identifier list mode. In identifier mask mode bits FACT(2n + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2n. 0 Filter x is deactivated. 1 Filter x is activated. This field can be set/cleared in Initialization mode only.

Identifier filter match index (IFMI)

Figure 439. Identifier filter match index (IFMI)

Address: Base + 0x0044 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	IFMI[0:4]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 416. IFMI field descriptions

Field	Description
0:26	Reserved
IFMI[0:4] 27:31	Filter match index This register contains the index corresponding to the received identifier. It can be used to directly write or read the data in SRAM (see *Section , Slave mode* for more details). When no filter matches, IFMI[0:4] = 0. When Filter *n* is matching, IFMI[0:4] = *n* + 1.

Identifier filter mode register (IFMR)

Figure 440. Identifier filter mode register (IFMR)

Offset: 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	IFM							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 417. IFMR field descriptions

Field	Description
IFM	Filter mode (see *Table 418*). 0 Filters 2*n* and 2*n* + 1 are in identifier list mode. 1 Filters 2*n* and 2*n* + 1 are in mask mode (filter 2*n* + 1 is the mask for the filter 2*n*).

Table 418. IFMR[IFM] configuration

Bit	Value	Result
IFM[0]	0	Filters 0 and 1 are in identifier list mode.
	1	Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).
IFM[1]	0	Filters 2 and 3 are in identifier list mode.
	1	Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).
IFM[2]	0	Filters 4 and 5 are in identifier list mode.
	1	Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).
IFM[3]	0	Filters 6 and 7 are in identifier list mode.
	1	Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).
IFM[4]	0	Filters 8 and 9 are in identifier list mode.
	1	Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).
IFM[5]	0	Filters 10 and 11 are in identifier list mode.
	1	Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10).
IFM[6]	0	Filters 12 and 13 are in identifier list mode.
	1	Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12).
IFM[7]	0	Filters 14 and 15 are in identifier list mode.
	1	Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14).

Identifier filter control register (IFCR2*n*)

Figure 441. Identifier filter control register (IFCR2*n*)

Offsets: 0x004C–0x0084 (8 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	DFL			DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Register bit can be read in any mode, written only in Initialization mode

Table 419. IFCR2*n* field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex transmits the data from the BDRL and BDRM registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity.

Identifier filter control register (IFCR2*n* + 1)

Figure 442. Identifier filter control register (IFCR2*n* + 1)

Offsets: 0x0050–0x0088 (8 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	DFL			DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Register bit can be read in any mode, written only in Initialization mode

Table 420. IFCR2*n* + 1 field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex transmits the data from the BDRL and BDRM registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data field only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity

23.8 Functional description

23.8.1 UART mode

The main features in the UART mode are

- Full duplex communication
- 8- or 9-bit data with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

8-bit data frames: The 8th bit can be a data or a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

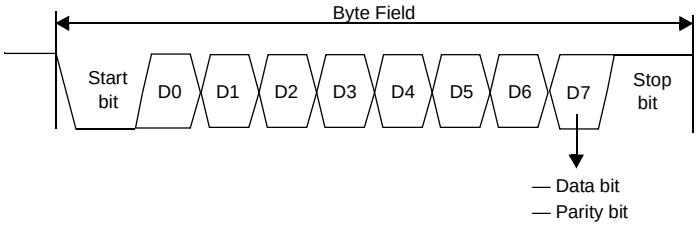


Figure 443. UART mode 8-bit data frame

9-bit frames: The 9th bit is a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case.

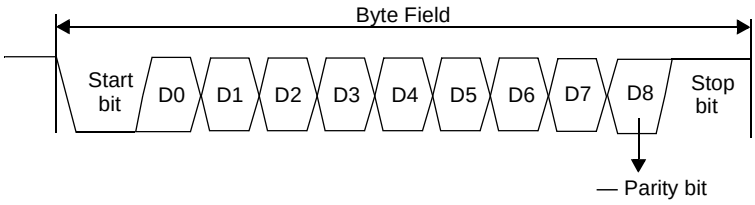


Figure 444. UART mode 9-bit data frame

Buffer in UART mode

The 8-byte buffer is divided into two parts: one for receiver and one for transmitter as shown in *Table 421*.

Table 421. Message buffer

Buffer data register	LIN mode		UART mode	
BDRL[0:31]	Transmit/Receive buffer	DATA0[0:7]	Transmit buffer	Tx0
		DATA1[0:7]		Tx1
		DATA2[0:7]		Tx2
		DATA3[0:7]		Tx3
BDRM[0:31]		DATA4[0:7]	Receive buffer	Rx0
		DATA5[0:7]		Rx1
		DATA6[0:7]		Rx2
		DATA7[0:7]		Rx3

UART transmitter

In order to start transmission in UART mode, you must program the UART bit and the transmitter enable (TXEN) bit in the UARTCR to 1. Transmission starts when DATA0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by UARTCR[TDFL] (see *Table 402*).

The Transmit buffer is 4 bytes, hence a 4-byte maximum transmission can be triggered. Once the programmed number of bytes has been transmitted, the UARTSR[DTF] bit is set. If UARTCR[TXEN] is reset during a transmission then the current transmission is completed and no further transmission can be invoked.

UART receiver

The UART receiver is active as soon as the user exits Initialization mode and programs UARTCR[RXEN] = 1. There is a dedicated 4-byte data buffer for received data bytes. Once the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRF] bit is set. If the RXEN bit is reset during a reception then the current reception is completed and no further reception can be invoked until RXEN is set.

If a parity error occurs during reception of any byte, then the corresponding PE*x* bit in the UARTSR is set. No interrupt is generated in this case. If a framing error occurs in any byte (UARTSR[FE] = 1) then an interrupt is generated if the LINIER[FEIE] bit is set.

If the last received frame has not been read from the buffer (that is, RMB bit is not reset by the user) then upon reception of the next byte an overrun error occurs (UARTSR[BOF] = 1) and one message will be lost. Which message is lost depends on the configuration of LINCR1[RBLM].

- If the buffer lock function is disabled (LINCR1[RBLM] = 0) the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

An interrupt is generated if the LINIER[BOIE] bit is set.

Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC_ME). In UART mode, the LINFlex controller acknowledges a clock gating request once the data transmission and data reception are completed, that is, once the Transmit buffer is empty and the Receive buffer is full.

23.8.2 LIN mode

LIN mode comprises four submodes:

- Master mode
- Slave mode
- Slave mode with identifier filtering
- Slave mode with automatic resynchronization

These submodes are described in the following pages.

Master mode

In Master mode the application uses the message buffer to handle the LIN messages. Master mode is selected when the LINCR1[MME] bit is set.

LIN header transmission

According to the LIN protocol any communication on the LIN bus is triggered by the Master sending a header. The header is transmitted by the Master task while the data is transmitted by the Slave task of a node.

To transmit a header with LINFlex the application must set up the identifier, the data field length and configure the message (direction and checksum type) in the BIDR before requesting the header transmission by setting LINCR2[HTRQ].

Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the application must provide the data to LINFlex before requesting the header transmission. The application stores the data in the message buffer BDR. According to the data field length, LINFlex transmits the data and the checksum. The application uses the BDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

If the response has been sent successfully, the LINSR[DTF] bit is set. In case of error, the DTF flag is not set and the corresponding error flag is set in the LINESR (see *Section , Error handling*).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBEF] bit is set after the first 8 bytes have been transmitted. The application has to update the buffer BDR before resetting the DBEF bit. The transmission of the next bytes starts when the DBEF bit is reset.

After the last data byte (or the checksum byte) has been sent, the DTF flag is set.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the application sets this bit the response is sent by LINFlex (publisher). Resetting this bit configures the message buffer as subscriber.

Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlex stores the data received from the slave in the message buffer and stores the message status in the LINSR.

If the response has been received successfully, the LINSR[DRF] is set. In case of error, the DRF flag is not set and the corresponding error flag is set in the LINESR (see *Section , Error handling*).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBFF] bit is set once the first 8 bytes have been received. The application has to read the buffer BDR before resetting the DBFF bit. Once the last data byte (or the checksum byte) has been received, the DRF flag is set.

Data discard

To discard data from a slave, the BIDR[DIR] bit must be reset and the LINCR2[DDRQ] bit must be set before starting the header transmission.

Error detection

LINFlex is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

In Master mode, the following errors are detected:

- **Bit error**: During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error**: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error**: The computed checksum does not match the received one.
- **Response and Frame timeout**: See *Section 23.8.3, 8-bit timeout counter*, for more details.

Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if LINIER[BEIE] = 1.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting

the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer is overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.

Slave mode

In Slave mode the application uses the message buffer to handle the LIN messages. Slave mode is selected when LINCR1[MME] = 0.

Data transmission (transceiver as publisher)

When LINFlex receives the identifier, the LINSR[HRF] is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR, fill the BDR registers, specify the data field length using the BIDR[DFL] and trigger the data transmission by setting the LINCR2[DTRQ] bit.

One or several identifier filters can be configured for transmission by setting the IFCR*x*[DIR] bit and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in transmission and activated, and if the received identifier matches the filter, a specific TX interrupt (instead of an RX interrupt) is generated.

Typically, the application has to copy the data from SRAM locations to the BDR. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data to the BDR (see *Figure 446*).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BIDR. The software fills the BDR and triggers the data transmission by programming LINCR2[DTRQ] = 1.

If LINFlex cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (see *Section , Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

Data reception (transceiver as subscriber)

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR and specify the data field length using the BIDR[DFL] field before receiving the stop bit of the first byte of data field.

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by programming IFCR*x*[DIR] = 0 and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in reception and activated, and if the received identifier matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the application has to copy the data from the BDR to SRAM locations. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data from the BDR to the SRAM (see *Figure 446*).

Using a filter avoids the software reading the ID value in the BIDR, and configuring the direction, the data field length and the checksum type in the BIDR.

If LINFlex cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (see *Section , Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

Data discard

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. If the received identifier does not concern the node, you must program LINCR2[DDRQ] = 1. LINFlex returns to idle state after bit DDRQ is set.

Error detection

In Slave mode, the following errors are detected:

- **Header error**: An error occurred during header reception (Break Delimiter error, Inconsistent Synch Field, Header Timeout).
- **Bit error**: During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error**: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error**: The computed checksum does not match the received one.

Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if the BEIE bit in the LINIER is set.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

During header reception, a Break Delimiter error, an Inconsistent Synch Field or a Timeout error leads LINFlex to discard the header. An interrupt is generated if LINIER[HEIE] = 1. LINFlex returns to idle state.

Valid header

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid Break Field and Break Delimiter come before the end of the current header or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

Valid message

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer will be overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.
- If buffer is not released (LINSR[RMB] = 1) before reception of next Identifier and if RBLM is set then ID along with the data is discarded.

Slave mode with identifier filtering

In the LIN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On header reception a slave node decides—depending on the identifier value—whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

To fulfill this requirement, the LINFlex controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources that would otherwise be needed by software for filtering.

Filter mode

Usually each of the sixteen IFCR registers filters one dedicated identifier, but this limits the number of identifiers LINFlex can handle to the number of IFCR registers implemented in the device. Therefore, in order to be able to handle more identifiers, the filters can be configured in mask mode.

In **identifier list mode** (the default mode), both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register.

In **mask mode**, the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”. For the bit mapping and registers organization, please see *Figure 445*.

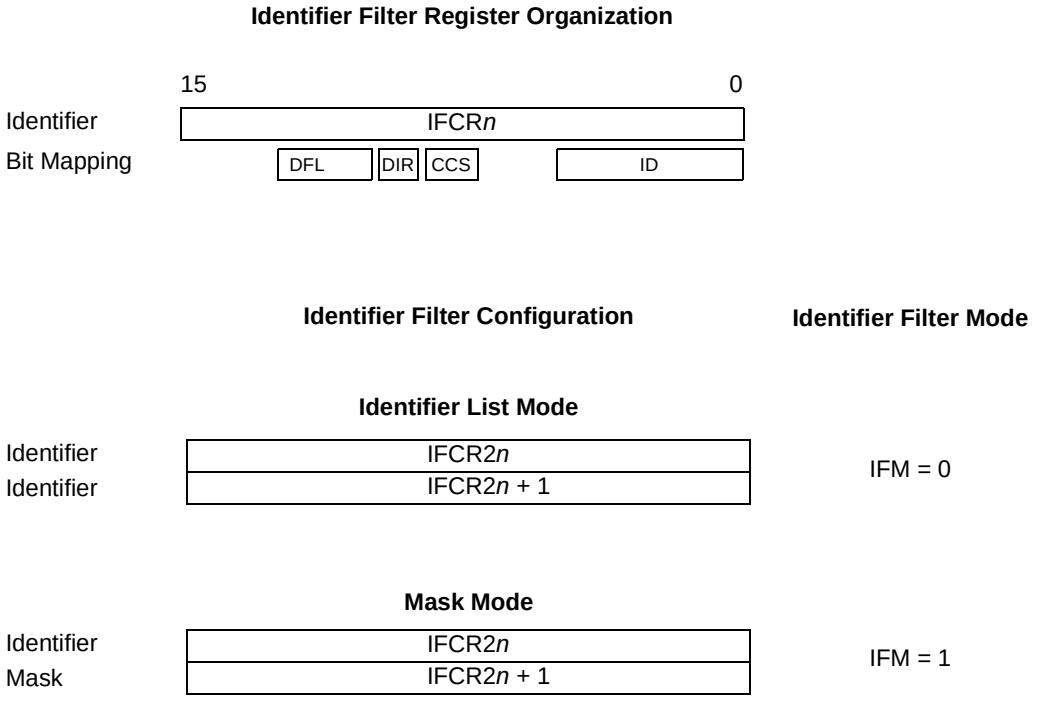


Figure 445. Filter configuration—register organization

Identifier filter mode configuration

The identifier filters are configured in the IFCR*x* registers. To configure an identifier filter the filter must first be activated by programming IFER[FACT] = 1. The **identifier list** or **identifier mask** mode for the corresponding IFCR*x* registers is configured by the IFMR[IFM] bit. For each filter, the IFCR*x* register configures the ID (or the mask), the direction (TX or RX), the data field length, and the checksum type.

If no filter is active, an RX interrupt is generated on any received identifier event.

If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

Table 422. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
0	0	0	RX interrupt on all identifiers

Table 422. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
a (a > 0)	a	0	— TX interrupt on identifiers matching the filters, — RX interrupt on all other identifiers if BF bit is set, no RX interrupt if BF bit is reset
n (n = a + b)	a (a > 0)	b (b > 0)	— TX interrupt on identifiers matching the TX filters, — RX interrupt on identifiers matching the RX filters, — all other identifiers discarded (no interrupt)
b (b > 0)	0	b	— RX interrupt on identifiers matching the filters, — TX interrupt on all other identifiers if BF bit is set, no TX interrupt if BF bit is reset

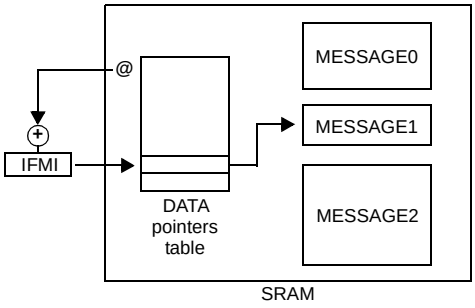


Figure 446. Identifier match index

Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in Slave mode if $f_{periph\_set\_1\_clk}$ tolerance is greater than 1.5%. This feature compensates a $f_{periph\_set\_1\_clk}$ deviation up to 14%, as specified in LIN standard.

This mode is similar to Slave mode as described in *Section , Slave mode*, with the addition of automatic resynchronization enabled by the LASE bit. In this mode LINFlex adjusts the fractional baud rate generator after each Synch Field reception.

Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN Break, the time duration between five falling edges on RDI is sampled on $f_{periph\_set\_1\_clk}$ and the result of this measurement is stored in an internal 19-bit register called SM (not user accessible) (see *Figure 447*). Then the LFDIV value (and its associated registers LINIBRR and LINFBRR) is automatically updated at the end of the fifth falling edge. During LIN Synch Field

measurement, the LINFlex state machine is stopped and no data is transferred to the data register.

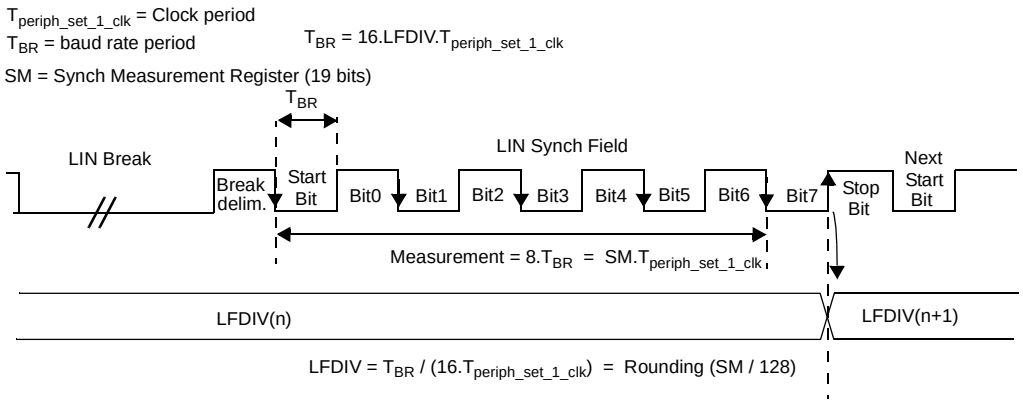


Figure 447. LIN synch field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 12 bits in the LINIBRR and the fraction is coded on 4 bits in the LINFBRR.

If LASE bit = 1 then LFDIV is automatically updated at the end of each LIN Synch Field.

Three internal registers (not user-accessible) manage the auto-update of the LINFlex divider (LFDIV):

- LFDIV_NOM (nominal value written by software at LINIBRR and LINFBRR addresses)
- LFDIV_MEAS (results of the Field Synch measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV_NOM.

Deviation error on the Synch Field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN Synch Field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the Synch Field:

- If D1 > 14.84%, LHE is set.
- If D1 < 14.06%, LHE is not set.
- If 14.06% < D1 < 14.84%, LHE can be either set or reset depending on the dephasing between the signal on LINFlex_RX pin the $f_{periph\_set\_1\_clk}$ clock.

The second check is based on a measurement of time between each falling edge of the Synch Field:

- If D2 > 18.75%, LHE is set.
- If D2 < 15.62%, LHE is not set.
- If 15.62% < D2 < 18.75%, LHE can be either set or reset depending on the dephasing between the signal on LINFlex_RX pin the $f_{periph\_set\_1\_clk}$ clock.

Note that the LINFlex does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC_ME). In LIN mode, the LINFlex controller acknowledges a clock gating request once the frame transmission or reception is completed.

23.8.3 8-bit timeout counter

LIN timeout mode

Resetting the LTOM bit in the LINTCSR enables the LIN timeout mode. The LINOCR becomes read-only, and OC1 and OC2 output compare values in the LINOCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the LINCR1[MME] bit), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BIDR is configured with a value higher than 8 data bytes).

LIN Master mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO = 28-bit time.

Field OC1 checks T_{Header} and $T_{Response}$ and field OC2 checks T_{Frame} (see *Figure 448*).

When LINFlex moves from Break delimiter state to Synch Field state (see *Section , LIN status register (LINSR)*):

- OC1 is updated with the value of OC_{Header} (OC_{Header} = CNT + 28),
- OC2 is updated with the value of OC_{Frame} (OC_{Frame} = CNT + 28 + RTO × 9 (frame timeout value for an 8-byte frame),
- the TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of $OC_{Response}$ ($OC_{Response}$ = CNT + RTO × 9 (response timeout value for an 8-byte frame)).

On the first response byte is received, OC1 and OC2 are automatically updated to check $T_{Response}$ and T_{Frame} according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response (DFL = 7) is always assumed.

LIN Slave mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.

OC1 checks T_{Header} and $T_{Response}$ and OC2 checks T_{Frame} (see *Figure 448*).

When LINFlex moves from Break state to Break Delimiter state (see *Section , LIN status register (LINSR)*):

- OC1 is updated with the value of OC_{Header} (OC_{Header} = CNT + HTO),
- OC2 is updated with the value of OC_{Frame} (OC_{Frame} = CNT + HTO + RTO × 9 (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of $OC_{Response}$ ($OC_{Response}$ = CNT + RTO × 9 (response timeout value for an 8-byte frame)).

Once the first response byte is received, OC1 and OC2 are automatically updated to check $T_{Response}$ and T_{Frame} according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is reset.

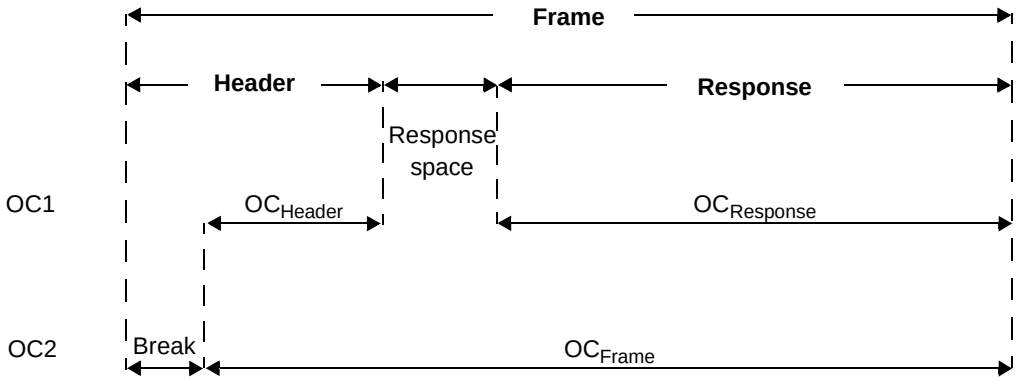


Figure 448. Header and response timeout

Output compare mode

Programming LINTCSR[LTOM] = 1 enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the LINTOCR by software.

23.8.4 Interrupts

Table 423. LINFlex interrupt control

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Header Received interrupt	HRF	HRIE	RXI (1)
Data Transmitted interrupt	DTF	DTIE	TXI
Data Received interrupt	DRF	DRIE	RXI
Data Buffer Empty interrupt	DBEF	DBEIE	TXI
Data Buffer Full interrupt	DBFF	DBFIE	RXI

Table 423. LINFlex interrupt control (continued)

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Wake-up interrupt	WUPF	WUPIE	RXI
LIN State interrupt (2)	LSF	LSIE	RXI
Buffer Overrun interrupt	BOF	BOIE	ERR
Framing Error interrupt	FEF	FEIE	ERR
Header Error interrupt	HEF	HEIE	ERR
Checksum Error interrupt	CEF	CEIE	ERR
Bit Error interrupt	BEF	BEIE	ERR
Output Compare interrupt	OCF	OCIE	ERR
Stuck at Zero interrupt	SZF	SZIE	ERR

1. In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.
2. For debug and validation purposes

24 FlexCAN

24.1 Introduction

The FlexCAN module is a communaication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in *Figure 449*, which describes the main subblocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask Registers. Support for 32 MBs is provided. The functions of the submodules are described in subsequent sections.

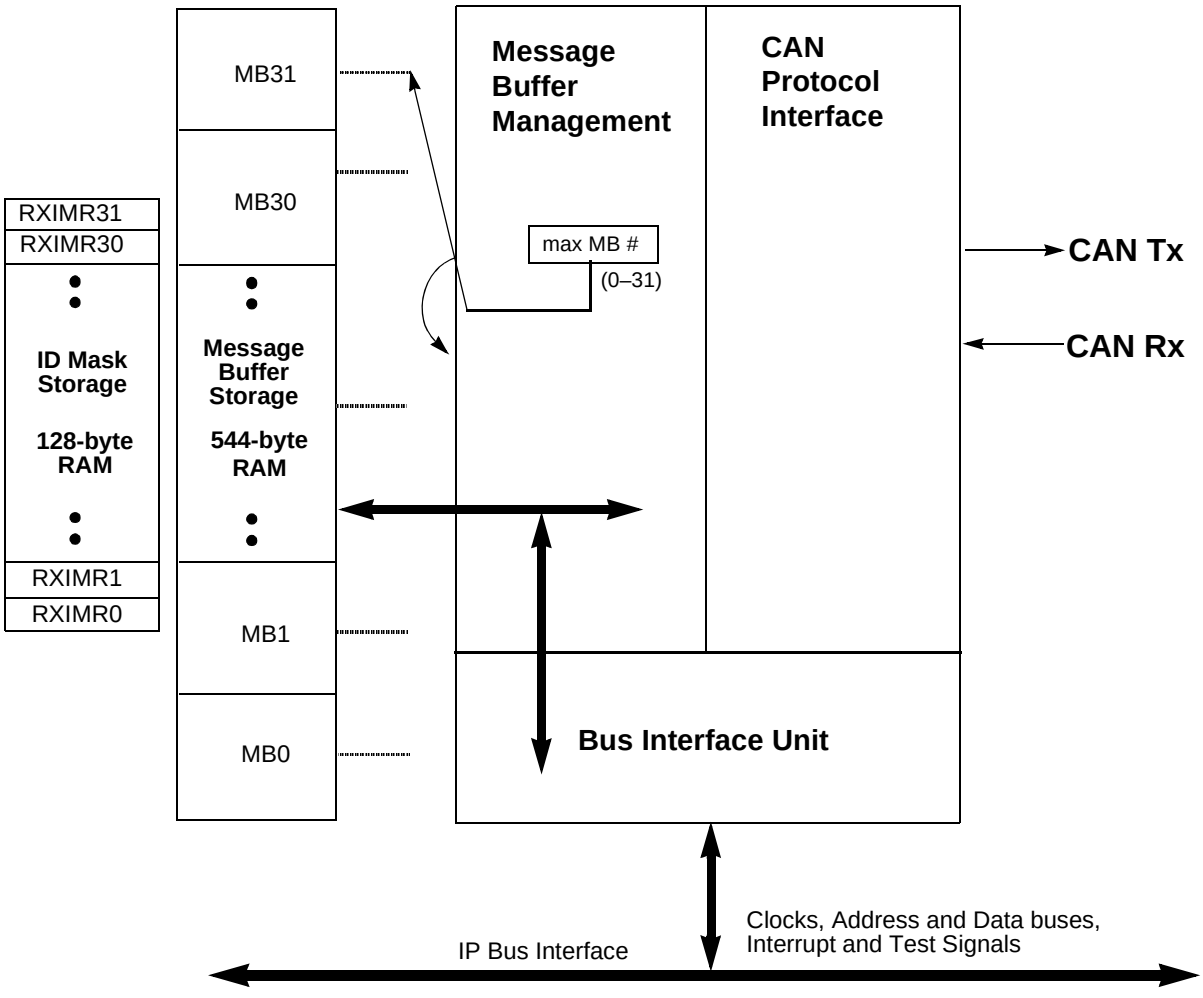


Figure 449. FlexCAN block diagram

24.1.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation

in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. 32 Message Buffers are supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) submodule manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) submodule handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) submodule controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

Three FlexCAN module are implemented on SPC56xP60/54 device. Two are named as FlexCAN_0 and FlexCAN_1 and one is Safetyport.

24.1.2 FlexCAN module features

The FlexCAN module includes these features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - 0 to 8 bytes data length
  - Programmable bit rate as high as 1 Mbit/s
  - Content-related addressing
- 32 Flexible Message Buffers (MBs) of 0 to 8 bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes 544 bytes (32 MBs) of RAM used for MB storage
- Includes 128 bytes (32 MBs) of RAM used for individual Rx Mask Registers
- Full-featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard, or 32 partial (8-bit) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen-only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts

- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes, with programmable wake up

24.1.3 Modes of operation

The FlexCAN module has four functional modes: Normal mode (User and Supervisor), Freeze mode, Listen-Only Mode, and Loop-Back mode. There are also these low power modes: Disable mode and Stop mode.

- Normal mode (User or Supervisor)

  In Normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor modes differ in the access to some restricted control registers.

- Freeze mode

  It is enabled when the FRZ bit in the Module Configuration Register (MCR) is asserted. If enabled, Freeze Mode is entered when the HALT bit in the MCR is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See *Section : Freeze mode* for more information.

- Listen-Only mode

  The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

- Loop-Back mode

  The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

- Module Disable mode

  This low power mode is entered when the MDIS bit in the MCR is asserted. When disabled, the module shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. Exit from this mode is done by negating the MDIS bit in the MCR. See *Section : Module disable mode* for more information.

- Stop mode

  This low power mode is entered when Stop mode is requested at device level. When in Stop mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop mode request is removed. See *Section : Stop mode* for more information.

24.2 External signal description

24.2.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 424* and described in more detail in the next subsections.

Table 424. FlexCAN signals

Signal name	Direction	Description
RXD	Input	CAN receive pin
TXD	Output	CAN transmit pin

24.2.2 Signal Descriptions

RXD

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

TXD

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

24.3 Memory map and registers description

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the device. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

24.3.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 32 MBs capability is shown in *Table 425*. The access type can be Supervisor (S) or Unrestricted (U), also called User access. Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the SUPV bit in the MCR.

The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK) and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in the MCR is asserted.

The address ranges 0x0060–0x027F and 0x0880–0x08FF are occupied by two separate embedded memories of RAM, of 544 bytes and 128 bytes, respectively. When the FlexCAN is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the

BCC bit in the MCR is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the address range 0x0880–0x097F is considered reserved space, independent of the value of the BCC bit.

Table 425. FlexCAN module memory map

Offset from FlexCAN_BASE 0xFFFC_0000 (CAN_0) 0x8FFC_4000 (CAN_1)	Register	Location
0x0000	Module Configuration Register (MCR)	*on page 777*
0x0004	Control Register (CTRL)	*on page 781*
0x0008	Free Running Timer (TIMER)	*on page 784*
0x000C	Reserved	
0x0010	Rx Global Mask (RXGMASK)	*on page 785*
0x0014	Rx Buffer 14 Mask (RX14MASK)	*on page 786*
0x0018	Rx Buffer 15 Mask (RX15MASK)	*on page 786*
0x001C	Error Counter Register (ECR)	*on page 787*
0x0020	Error and Status Register (ESR)	*on page 788*
0x0024	Reserved	
0x0028	Interrupt Masks 1 (IMASK1)	*on page 791*
0x002C	Reserved	
0x0030	Interrupt Flags 1 (IFLAG1)	*on page 792*
0x0034–0x005F	Reserved	
0x0060–0x007F	Serial Message Buffers (SMB0–SMB1) – Reserved	—
0x0080–0x017F	Message Buffers MB0–MB15	*on page 771*
0x0180–0x027F	Message Buffers MB16–MB31	*on page 771*
0x0280–0x087F	Reserved	
0x0880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	*on page 793*
0x08C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	*on page 793*
0x0900–0x3FFF	Reserved	

Table 426. FlexCAN register reset status

Register	Affected by hard reset	Affected by soft reset
Module Configuration Register (MCR)	Yes	Yes
Control Register (CTRL)	Yes	No
Free Running Timer (TIMER)	Yes	Yes

Table 426. FlexCAN register reset status (continued)

Register	Affected by hard reset	Affected by soft reset
Reserved		
Rx Global Mask (RXGMASK)	Yes	No
Rx Buffer 14 Mask (RX14MASK)	Yes	No
Rx Buffer 15 Mask (RX15MASK)	Yes	No
Error Counter Register (ECR)	Yes	Yes
Error and Status Register (ESR)	Yes	Yes
Interrupt Masks 1 (IMASK1)	Yes	Yes
Interrupt Flags 1 (IFLAG1)	Yes	Yes
Serial Message Buffers (SMB0–SMB1) – Reserved	No	No
Message Buffers MB0–MB15	No	No
Message Buffers MB16–MB31	No	No
Rx Individual Mask Registers RXIMR0–RXIMR15	No	No
Rx Individual Mask Registers RXIMR16–RXIMR31	No	No

The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual MB is formed by 16 bytes mapped on memory as described in *Table 427*. The module also implements two additional Message Buffers called SMB0 and SMB1 (Serial Message Buffers), in the address ranges 0x60–0x6F and 0x70–0x7F, which are not accessible to the end user. They are used for temporary storage of frames during the reception and transmission processes. *Table 427* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 427. Message Buffer MB0 memory mapping

Address offset	MB field
0x0080	Control and Status (C/S)
0x0084	Identifier Field
0x0088–0x008F	Data Field 0 – Data Field 7 (1 byte each)

24.3.2 Message buffer structure

The Message Buffer structure used by the FlexCAN module is represented in *Table 450*. Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

Figure 450. Message buffer structure

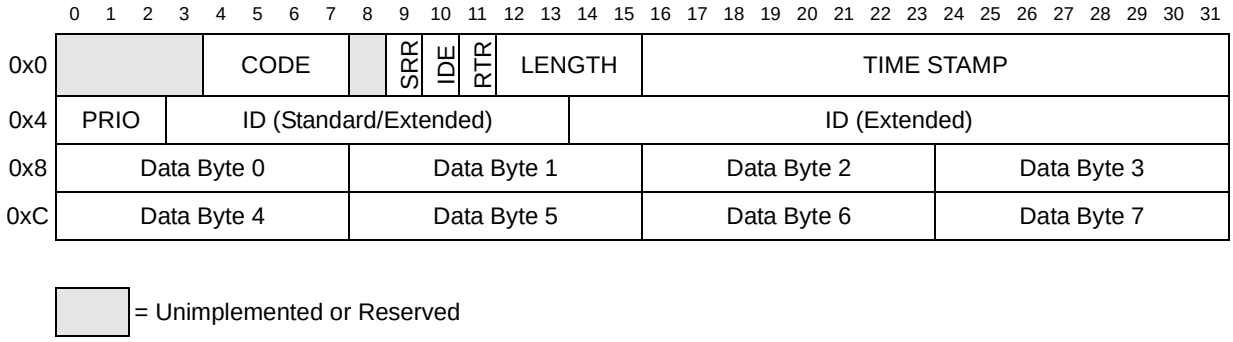


Table 428. Message Buffer structure field description

Field	Description
CODE	Message Buffer Code This 4-bit field can be accessed (read or write) by the CPU and by the Flexcan module itself, as part of the message buffer matching and arbitration process. The encoding is shown in *Table 429* and *Table 430*. See *Section 24.4: Functional description* for additional information.
SRR	Substitute Remote Request Fixed recessive bit, used only in extended format. It must be set to 1 by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in Extended Format frames. 1 Recessive value is compulsory for transmission in Extended Format frames.
IDE	ID Extended Bit This bit identifies whether the frame format is standard or extended. 0 Frame format is standard 1 Frame format is extended
RTR	Remote Transmission Request This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a Data Frame to be transmitted 1 Indicates the current MB has a Remote Frame to be transmitted
LENGTH	Length of Data in Bytes This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see *Table 450*). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.
TIME STAMP	Free-Running Counter Time Stamp This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.

Table 428. Message Buffer structure field description (continued)

Field	Description
PRIO	Local priority This 3-bit field is only used when LPRIO_EN bit is set in MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See *Section 24.4.3: Arbitration process.*
ID	Frame Identifier In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field As many as 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 429. Message buffer code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active.	–	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to *Section 24.4.5: Matching process* for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to *Section 24.4.5: Matching process* for details about overrun behavior.

Table 429. Message buffer code for Rx buffers (continued)

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0XY1[1]	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1. Note that for Tx MBs (see *Table 430*), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR.

Table 430. Message Buffer code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	–	INACTIVE: MB does not participate in the arbitration process.
X	1001	–	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

Table 431. MB0–MB31 addresses

Address	Register	Address	Register
Base + 0x0080	MB0	Base + 0x0180	MB16
Base + 0x0090	MB1	Base + 0x0190	MB17
Base + 0x00A0	MB2	Base + 0x01A0	MB18
Base + 0x00B0	MB3	Base + 0x01B0	MB19
Base + 0x00C0	MB4	Base + 0x01C0	MB20

Table 431. MB0–MB31 addresses (continued)

Address	Register	Address	Register
Base + 0x00D0	MB5	Base + 0x01D0	MB21
Base + 0x00E0	MB6	Base + 0x01E0	MB22
Base + 0x00F0	MB7	Base + 0x01F0	MB23
Base + 0x0100	MB8	Base + 0x0200	MB24
Base + 0x0110	MB9	Base + 0x0210	MB25
Base + 0x0120	MB10	Base + 0x0220	MB26
Base + 0x0130	MB11	Base + 0x0230	MB27
Base + 0x0140	MB12	Base + 0x0240	MB28
Base + 0x0150	MB13	Base + 0x0250	MB29
Base + 0x0160	MB14	Base + 0x0260	MB30
Base + 0x0170	MB15	Base + 0x0270	MB31

24.3.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFF (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *Figure 451* shows the Rx FIFO data structure. The region 0x80–0x8F contains an MB structure that is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDF is reserved for internal use of the FIFO engine. The region 0xE0–0xFF contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. *Table 432* shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See *Section 24.4.7: Rx FIFO* for more information.

Figure 451. Rx FIFO structure

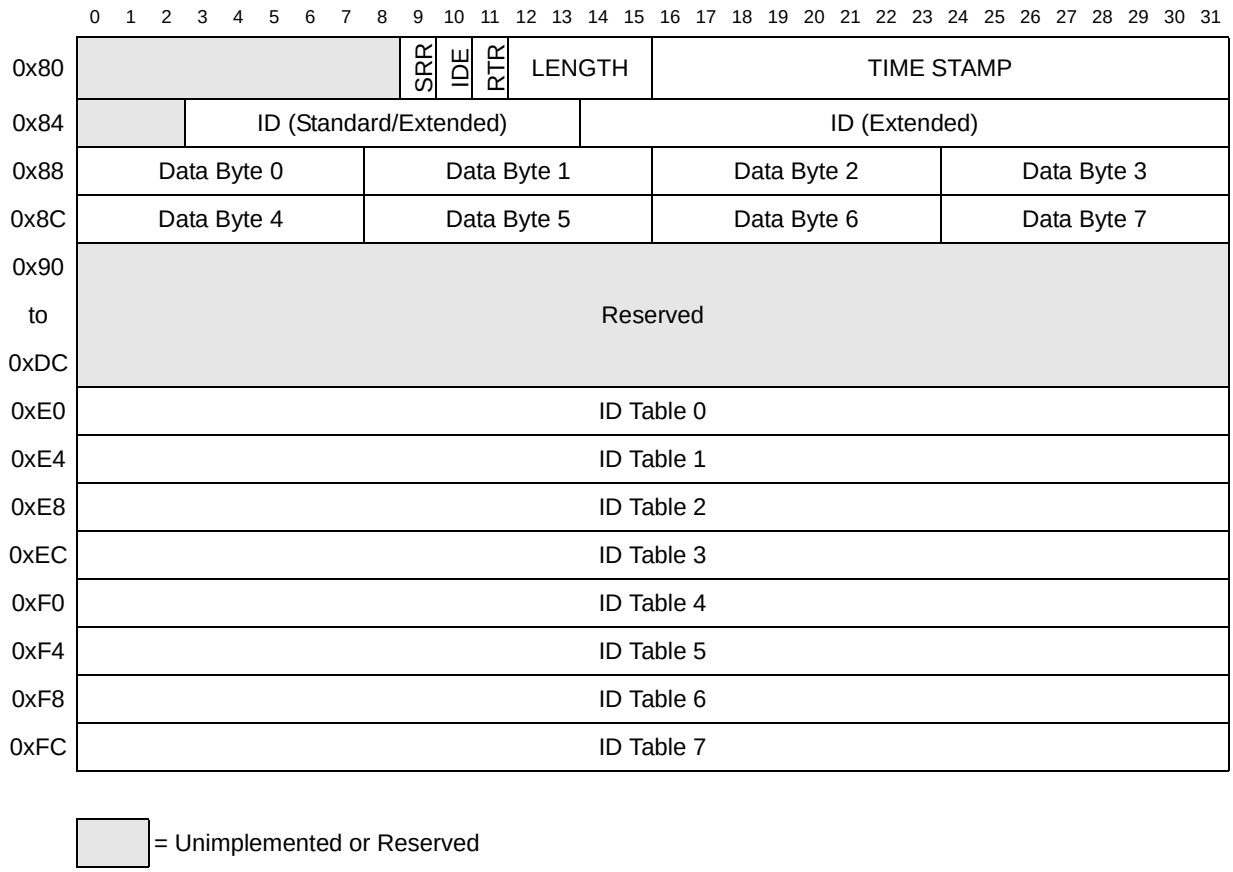


Table 432. ID Table 0 - 7

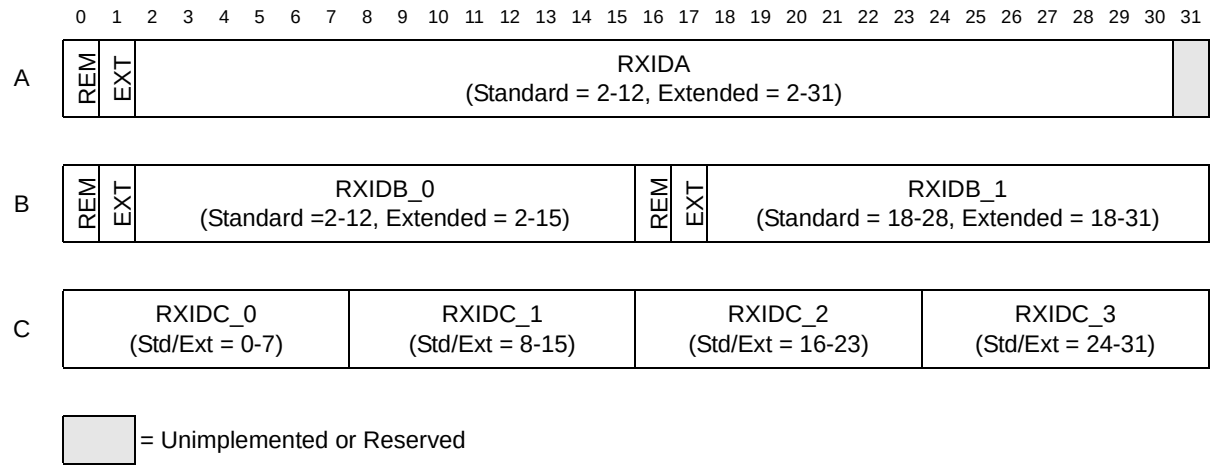


Table 433. Rx FIFO Structure field description

Field	Description
REM	Remote Frame This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 0 Remote Frames are rejected and data frames can be accepted. 1 Remote Frames can be accepted and data frames are rejected.
EXT	Extended Frame Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 0 Extended frames are rejected and standard frames can be accepted. 1 Extended frames can be accepted and standard frames are rejected.
RXIDA	Rx Frame Identifier (Format A) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0 RXIDB_1	Rx Frame Identifier (Format B) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0 RXIDC_1 RXIDC_2 RXIDC_3	Rx Frame Identifier (Format C) Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

24.3.4 Registers description

The FlexCAN registers are described in this section in ascending address order.

Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Figure 452. Module Configuration Register (MCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	SUPV	0	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W																
Reset	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Table 434. MCR field descriptions

Field	Description
0 MDIS	Module Disable This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. This is the only bit in MCR not affected by soft reset. See *Section : Module disable mode* for more information. 0 Enable the FlexCAN module. 1 Disable the FlexCAN module.
1 FRZ	Freeze Enable The FRZ bit specifies the FlexCAN behavior when the HALT bit in the MCR is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is able to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 0 Not able to enter Freeze Mode. 1 Able to enter Freeze Mode.
2 FEN	FIFO Enable This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See *Section 24.3.3: Rx FIFO structure* and *Section 24.4.7: Rx FIFO* for more information. 0 FIFO disabled. 1 FIFO enabled.
3 HALT	Halt FlexCAN Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See *Section : Freeze mode* for more information. 0 No Freeze Mode request. 1 Enters Freeze Mode if the FRZ bit is asserted.
4 NOT_RDY	FlexCAN Not Ready This read-only bit indicates that FlexCAN is either in Disable Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 0 FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode. 1 FlexCAN module is either in Disable Mode, Stop Mode or Freeze Mode.

Table 434. MCR field descriptions (continued)

Field	Description
6 SOFT_RST	Soft Reset When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IFLAG1. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: – CTRL – RXIMR0–RXIMR31 – RXGMASK, RX14MASK, RX15MASK – all Message Buffers The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied. 0 No reset request. 1 Resets the registers marked as “affected by soft reset” in *Table 425*.
7 FRZ_ACK	Freeze Mode Acknowledge This read-only bit indicates that FlexCAN is in Freeze mode and its prescaler is stopped. The Freeze mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See *Section : Freeze mode* for more information. 0 FlexCAN not in Freeze mode, prescaler running. 1 FlexCAN in Freeze mode, prescaler stopped.
8 SUPV	Supervisor Mode This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of *Table 425*. The reset value of this bit is 1, so the affected registers start with Supervisor access restrictions. 0 Affected registers are in Unrestricted memory space. 1 Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location.
10 WRN_EN	Warning Interrupt Enable When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. 0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters. 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <96 to ≥ 96.

Table 434. MCR field descriptions (continued)

Field	Description
11 LPM_ACK	Low Power Mode Acknowledge This read-only bit indicates that FlexCAN is either in Disable Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See *Section : Module disable mode* and *Section : Stop mode* for more information. 0 FlexCAN not in any of the low power modes. 1 FlexCAN is either in Disable Mode or Stop mode.
14 SRX_DIS	Self Reception Disable This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. 0 Self reception enabled 1 Self reception disabled
15 BCC	Backwards Compatibility Configuration This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: – For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK and RX15MASK. – The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to ‘0110’ (overrun). Upon reset this bit is negated, allowing legacy software to work without modification. 0 Individual Rx masking and queue feature are disabled. 1 Individual Rx masking and queue feature are enabled.
18 LPRIO_EN	Local Priority Enable This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It extends the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. 0 Local Priority disabled 1 Local Priority enabled
19 AEN	Abort Enable This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 0 Abort disabled 1 Abort enabled

Table 434. MCR field descriptions (continued)

Field	Description
22–23 IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 435*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 24.3.3: Rx FIFO structure.*
26–31 MAXMB	Maximum Number of Message Buffers This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field should be changed only while the module is in Freeze Mode. **Maximum MBs in use = MAXMB + 1** Note: MAXMB has to be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN will not transmit or receive frames.

Table 435. IDAM coding

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element.
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
10	C	Four partial 8-bit IDs (standard or extended) per filter element.
11	D	All frames rejected.

Control Register (CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK and BOFF_REC bits, that can be accessed at any time.

Figure 453. Control Register (CTRL)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 436. CTRL field descriptions

Field	Description
0–7 PRESDIV	Prescaler Division Factor This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to *Section : Protocol timing.* **Sclock frequency = CPI clock frequency / (PRESDIV + 1).**
8–9 RJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta(1) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. **Resync Jump Width = RJW + 1.**
10–12 PSEG1	PSEG1 — Phase Segment 1 This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. **Phase Buffer Segment 1(PSEG1 + 1) x Time-Quanta.**
13–15 PSEG2	PSEG2 — Phase Segment 2 This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. **Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.**
16 BOFF_MSK	Bus Off Mask This bit provides a mask for the Bus Off Interrupt. 0 Bus Off interrupt disabled. 1 Bus Off interrupt enabled.
17 ERR_MSK	Error Mask This bit provides a mask for the Error Interrupt. 0 Error interrupt disabled. 1 Error interrupt enabled.
18 CLK_SRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See *Section : Protocol timing* for more information. 0 The CAN engine clock source is the oscillator clock. 1 The CAN engine clock source is the bus clock. Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation.

Table 436. CTRL field descriptions (continued)

Field	Description
19 TWRN_MSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated. 0 Tx Warning Interrupt disabled. 1 Tx Warning Interrupt enabled.
20 RWRN_MSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated. 0 Rx Warning Interrupt disabled. 1 Rx Warning Interrupt enabled.
21 LPB	Loop Back This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 0 Loop Back disabled. 1 Loop Back enabled.
24 SMP	Sampling Mode This bit defines the sampling mode of CAN bits at the Rx input. 0 Just one sample determines the bit value. 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and two preceding samples, a majority rule is used.
25 BOFF_REC	Bus Off Recovery Mode This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will resynchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery. 0 Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B. 1 Automatic recovering from Bus Off state disabled.
26 TSYN	Timer Sync Mode This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (that is, global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 0 Timer Sync feature disabled 1 Timer Sync feature enabled

Table 436. CTRL field descriptions (continued)

Field	Description
27 LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the LPRIO_EN bit does not affect the priority arbitration. 0 Buffer with highest priority is transmitted first. 1 Lowest number buffer is transmitted first.
28 LOM	Listen-Only Mode This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 Listen Only Mode is deactivated. 1 FlexCAN module operates in Listen Only Mode.
29–31 PROPSEG	Propagation Segment This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) × Time-Quanta. Time-Quantum = one Sclock period.

1. One time quantum is equal to the Sclock period.

Free Running Timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Figure 454. Free Running Timer (TIMER)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TIMER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 437. TIMER field descriptions

Field	Description
TIMER	Holds the value for this timer.

Rx Global Mask register (RXGMASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in the MCR causes the RXGMASK register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

RXGMASK is used as an acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in the MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Figure 455. Rx Global Mask register (RXGMASK)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 438. RXGMASK field description

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

Rx 14 Mask (RX14MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in the MCR causes the RX14MASK register to have no effect on the module operation.

RX14MASK is used as an acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in the MCR is set (FIFO enabled), the RX14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Figure 456. Rx Buffer 14 Mask register (RX14MASK)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 439. RX14MASK field description

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

Rx 15 Mask (RX15MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in the MCR causes the RX15MASK register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in the MCR is set (FIFO enabled), the RX15MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Figure 457. Rx Buffer 15 Mask register (RX15MASK)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 440. RX15MASK field description

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

Error Counter Register (ECR)

This register has two 8-bit fields that reflect the value of two FlexCAN error counters:

- Transmit Error Counter (Tx_Err_Counter field)
- Receive Error Counter (Rx_Err_Counter field)

The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except inFreeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g., transmit ‘Error Active’ or ‘Error Passive’ flag, delay its transmission start time (‘Error Passive’) and avoid any

influence on the bus when in ‘Bus Off’ state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of Tx_Err_Counter or Rx_Err_Counter increases to be greater than or equal to 128, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Passive’ state.
- If the FlexCAN state is ‘Error Passive’, and either Tx_Err_Counter or Rx_Err_Counter decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to 0.
- If FlexCAN is in ‘Bus Off’ state, then Tx_Err_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, Tx_Err_Counter is reset to 0 and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the Tx_Err_Counter. When Tx_Err_Counter reaches the value of 128, the FLT_CONF field in the Error and Status Register is updated to be ‘Error Active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the Tx_Err_Counter value.
- If during system start-up, only one node is operating, then its Tx_Err_Counter increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK_ERR bit in the Error and Status Register). After the transition to ‘Error Passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.
- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Figure 458. Error Counter Register (ECR)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Rx_Err_Counter								Tx_Err_Counter							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16–21. Bits 22–28 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT and ERR_INT, that are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect). See *Section 24.4.10: Interrupts* for more details.

Figure 459. Error and Status Register (ESR)

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF		0	BOFF_INT	ERR_INT	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 441. Error and Status Register (ESR) field description

Field	Description
14 TWRN_INT	Tx Warning Interrupt Flag If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 The Tx error counter transitioned from < 96 to ≥ 96.
15 RWRN_INT	Rx Warning Interrupt Flag If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from 0 to 1, meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 The Rx error counter transitioned from < 96 to ≥ 96.
16 BIT1_ERR	Bit1 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence. 1 At least one bit sent as recessive is received as dominant. Note: This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
17 BIT0_ERR	Bit0 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence. 1 At least one bit sent as dominant is received as recessive.

Table 441. Error and Status Register (ESR) field description (continued)

Field	Description
18 ACK_ERR	Acknowledge Error This bit indicates that an Acknowledge Error has been detected by the transmitter node, that is, a dominant bit has not been detected during the ACK SLOT. 0 No such occurrence. 1 An ACK error occurred since last read of this register
19 CRC_ERR	Cyclic Redundancy Check Error This bit indicates that a CRC Error has been detected by the receiver node, that is, the calculated CRC is different from the received. 0 No such occurrence. 1 A CRC error occurred since last read of this register.
20 FRM_ERR	Form Error This bit indicates that a Form Error has been detected by the receiver node, that is, a fixed-form bit field contains at least one illegal bit. 0 No such occurrence. 1 A Form Error occurred since last read of this register.
21 STF_ERR	Stuffing Error This bit indicates that a Stuffing Error has been detected. 0 No such occurrence. 1 A Stuffing Error occurred since last read of this register.
22 TX_WRN	TX Error Counter This bit indicates when repetitive errors are occurring during message transmission. 0 No such occurrence. 1 TX_Err_Counter ≥ 96.
23 RX_WRN	Rx Error Counter This bit indicates when repetitive errors are occurring during message reception. 0 No such occurrence. 1 Rx_Err_Counter ≥ 96.
24 IDLE	CAN bus IDLE state This bit indicates when the CAN bus is in IDLE state. 0 No such occurrence. 1 CAN bus is now IDLE.
25 TXRX	Current FlexCAN status (transmitting/receiving) This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE = 0). 1 FlexCAN is transmitting a message (IDLE = 0).
26–27 FLT_CONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in *Table 442*. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted.

Table 441. Error and Status Register (ESR) field description (continued)

Field	Description
29 BOFF_INT	Bus Off Interrupt This bit is set when FlexCAN enters 'Bus Off' state. If the corresponding mask bit in the Control Register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 FlexCAN module entered 'Bus Off' state.
30 ERR_INT	Error Interrupt This bit indicates that at least one of the Error Bits (bits 16–21) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1.Writing 0 has no effect. 0 No such occurrence. 1 Indicates setting of any Error Bit in the Error and Status Register.

Table 442. Fault confinement state

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

Interrupt Masks 1 Register (IMASK1)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding IFLAG1 bit is set).

Figure 460. Interrupt Masks 1 Register (IMASK1)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 443. IMASK1 field descriptions

Field	Description
0–31 BUF31M – BUF0M	BUF31M–BUF0M — Buffer MB_i Mask Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 0 The corresponding buffer Interrupt is disabled. 1The corresponding buffer Interrupt is enabled. Note: Setting or clearing a bit in the IMASK1 Register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.

Interrupt Flags 1 Register (IFLAG1)

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to 1. Writing 0 has no effect.

When the AEN bit in the MCR is set (Abort enabled), while the IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the FEN bit in the MCR is set (FIFO enabled), the function of the eight least significant interrupt flags (BUF7I – BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Figure 461. Interrupt Flags 1 Register (IFLAG1)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 444. IFLAG1 field descriptions

Field	Description
0–23 BUF31I – BUF8I	Buffer MB_i Interrupt Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt. 0 No such occurrence. 1 The corresponding MB has successfully completed transmission or reception.
24 BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 0 No such occurrence. 1 MB7 completed transmission/reception or FIFO overflow.
25 BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 4 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 0 No such occurrence. 1 MB6 completed transmission/reception or FIFO almost full.
26 BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 0 No such occurrence. 1 MB5 completed transmission/reception or frames available in the FIFO.
27–31 BUF4I – BUF0I	Buffer MB_i Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 0 No such occurrence. 1 Corresponding MB completed transmission/reception.

Rx Individual Mask Registers (RXIMR0–RXIMR31)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled (FEN bit in MCR is set), the first 8 Mask Registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the MCR is negated, any read or write operation to these registers results in access error.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK and RX15MASK registers are available, regardless of the value of the BCC bit.

Figure 462. Rx Individual Mask Registers (RXIMR0–RXIMR31)

Address: See *Table 446* Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 445. RXIMR0–RXIMR31 field descriptions

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

Table 446. RXIMR0–RXIMR31 addresses

Address	Register	Address	Register
Base + 0x0880	RXIMR0	Base + 0x08C0	RXIMR16
Base + 0x0884	RXIMR1	Base + 0x08C4	RXIMR17
Base + 0x0888	RXIMR2	Base + 0x08C8	RXIMR18
Base + 0x088C	RXIMR3	Base + 0x08CC	RXIMR19
Base + 0x0890	RXIMR4	Base + 0x08D0	RXIMR20
Base + 0x0894	RXIMR5	Base + 0x08D4	RXIMR21
Base + 0x0898	RXIMR6	Base + 0x08D8	RXIMR22
Base + 0x089C	RXIMR7	Base + 0x08DC	RXIMR23
Base + 0x08A0	RXIMR8	Base + 0x08E0	RXIMR24
Base + 0x08A4	RXIMR9	Base + 0x08E4	RXIMR25
Base + 0x08A8	RXIMR10	Base + 0x08E8	RXIMR26
Base + 0x08AC	RXIMR11	Base + 0x08EC	RXIMR27
Base + 0x08B0	RXIMR12	Base + 0x08F0	RXIMR28
Base + 0x08B4	RXIMR13	Base + 0x08F4	RXIMR29
Base + 0x08B8	RXIMR14	Base + 0x08F8	RXIMR30
Base + 0x08BC	RXIMR15	Base + 0x08FC	RXIMR31

24.4 Functional description

24.4.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of as many as 32 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 24.3.2: Message buffer structure*). The memory corresponding to the first eight Message Buffers can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (as many as 8 extended IDs or 16 standard IDs or 32 eight-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a 0000 code is inactive (refer to *Table 429*). Similarly, a Tx MB with a 1000 or 1001 code is also inactive (refer to *Table 430*). An MB not programmed with 0000, 1000, or 1001 will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see *Section : Message Buffer deactivation*).

24.4.2 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

1. If the MB is active (transmission pending), write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see *Section : Transmission abort mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see *Section : Message Buffer deactivation*).
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step 4 (see *Table 429* and *Table 430* in *Section 24.3.2: Message buffer structure*). When the Abort feature is enabled (AEN in MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update

it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

24.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID[i] or the lowest MB number or the highest priority, depending on the LBUF and LPRIO_EN bits on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze Mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in the MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits as many as 8 data bytes, even if the DLC (Data Length Code) value is bigger.

24.4.4 Receive process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

1. If the MB has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted

i. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

(see *Section : Transmission abort mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write '1000' to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see *Section : Message Buffer deactivation*). If the MB already programmed as a receiver, just write '0000' to the Code field of the Control and Status word to keep the MB inactive.

2. Write the ID word
3. Write '0100' to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see *Table 429* and *Table 430* in *Section 24.3.2: Message buffer structure*)
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 24.4.6: Data coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in *Table 429*. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: **never perform polling by directly reading the C/S word of the MBs. Instead, read the IFLAG registers**.

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the MCR is not asserted. If SRX_DIS is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see *Section 24.4.7: Rx FIFO*). Upon receiving the frames

available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

24.4.5 Matching process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see *Section : Message Buffer lock mechanism*)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to *Table 429* and *Table 430*). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see *Section : Message Buffer lock mechanism*).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in the MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to *Section : Rx Individual Mask Registers (RXIMR0–RXIMR31).* During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR is negated.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK, and RX15MASK registers are available, regardless of the value of the BCC bit.

24.4.6 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Transmit process and *Section 24.4.4: Receive process.* Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

Transmission abort mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the Code field, the interrupt flag is set in the IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

1. CPU writes 1001 into the code field of the C/S word.
2. CPU reads the CODE field and compares it to the value that was written.
3. If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE = 1001) or it was transmitted (CODE = 1000).

Message Buffer deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. In order to avoid this situation, the abort procedures described in *Section : Transmission abort mechanism* should be used.

Message Buffer lock mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

Note: The locking mechanism only applies to Rx MBs that have a code different than INACTIVE (‘0000’) or EMPTY[j] (‘0100’). Also, Tx MBs can not be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

Note: If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

24.4.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see *Section 24.3.3: Rx FIFO structure*). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store as many as six frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80

j. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. In current FlexCAN versions, this behavior is maintained when the BCC bit is negated.

with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 4frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8 32-bit registers that can be configured to one of the following formats (see also *Section 24.3.3: Rx FIFO structure*):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

Note: A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0–RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

24.4.8 CAN protocol related features

Remote frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to 1. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field ‘1010’. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), FlexCAN will not generate an automatic response for Remote Request Frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

Overload frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

Time stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in *Section : Control Register (CTRL).*

Protocol timing

Figure 463 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK_SRC) in the CTRL Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

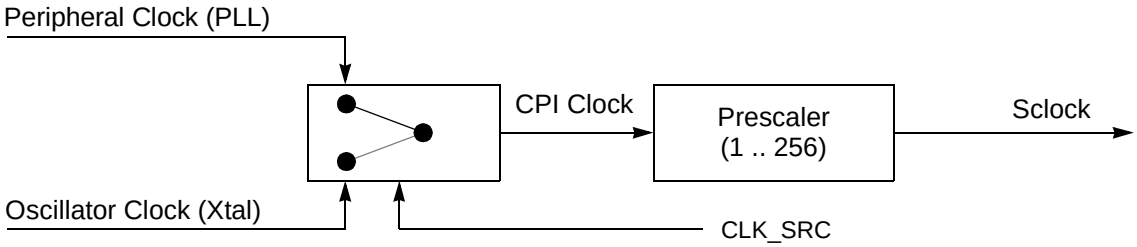


Figure 463. CAN engine clocking scheme

The crystal oscillator clock should be selected whenever a tight tolerance (to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, the CLK_SRC bit in the CTRL Register has no effect on the module operation.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See *Section : Control Register (CTRL).*

The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the 'time quantum' used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

Equation 66

$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler\ \ Value)}$$

A bit time is subdivided into three segments[k] (reference *Figure 464* and *Table 447*):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL Register (plus 1) to be 2 to 8 time quanta long.

Equation 67

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

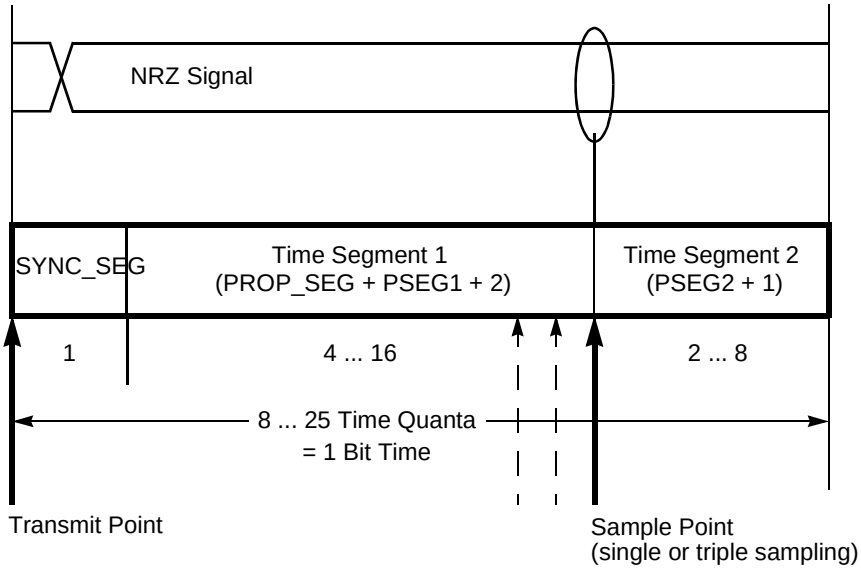


Figure 464. Segments within the bit time

k. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 447. Time segment syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 448 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 448. CAN standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Resynchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

Note: It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

Arbitration and matching timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in *Figure 465*.

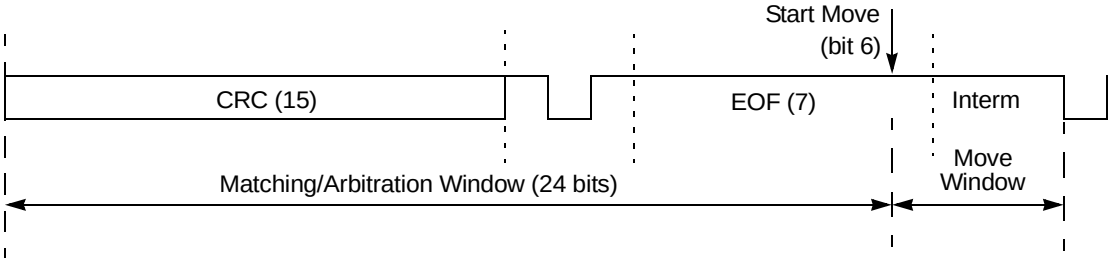


Figure 465. Arbitration, match, and move time windows

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in *Table 448*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, that is, the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in *Table 449*

Table 449. Minimum ratio between peripheral clock frequency and CAN bit rate

Number of message buffers	Minimum ratio
16	8
32	8

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in *Table 449* can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 32 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to 1 and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

24.4.9 Modes of operation details

Freeze mode

This mode is entered by asserting the HALT bit in the MCR or when the MCU is put into Debug Mode. In both cases it is also necessary that the FRZ bit is asserted in the MCR and the module is not in any of the low power modes. When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in MCR

After requesting Freeze Mode, the user must wait for the FRZ_ACK bit to be asserted in the MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates the FRZ bit in the MCR
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

Module disable mode

This low power mode is entered when the MDIS bit in the MCR is asserted. If the module is disabled during Freeze Mode, it shuts down the clocks to the CPI and MBM sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register, and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM_ACK bit.

Stop mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets the LPM_ACK bit, negates the FRZ_ACK bit and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and LPM_ACK bits in MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Stop Mode is exited by the CPU resuming the clocks and removing the Stop Mode request.

24.4.10 Interrupts

The module can generate as many as 38 interrupt sources (32 interrupts due to message buffers and 6 interrupts due to ORed interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning, and Wake Up[l]). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the

l. Wakeup interrupt never occurs because self-wakeup mechanism is not implemented.

buffers has assigned a flag bit in the IFLAG Registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1 (unless another interrupt is generated at the same time).

Note: It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags that are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (bit FEN on MCR set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFLAG1 becomes the FIFO Overflow flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the Frames Available in FIFO flag and bits 4:0 are unused. See *Section : Interrupt Flags 1 Register (IFLAG1)* for more information.

A combined interrupt for all MBs is also generated by an OR of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG Registers to determine which MB caused the interrupt.

The other five interrupt sources (Bus Off, Error, Tx Warning, Rx Warning, and Wakeup$^{(l)}$) generate interrupts like the MB ones, and can be read from the ESR register. The Bus Off, Error, Tx Warning, and Rx Warning interrupt mask bits are located in the CTRL register, and the Wake-Up interrupt mask bit is located in the MCR.

24.4.11 Bus interface

The CPU access to FlexCAN registers is subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the BCC bit in the MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 32 MBs and MAXMB is programmed with 0. The maximum number of MBs in this case becomes 1. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x027F. The available memory in the Mask Registers space would be from 0x0884 to 0x08FF.

Note: Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

24.5 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

24.5.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- SOFT_RST bit in MCR, which has the same effect as the MCU level soft reset

The clock source (CLK_SRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is unsynchronized to the CAN bus, the HALT and FRZ bits in the MCR are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the MCR are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see *Section : Freeze mode*). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN_EN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in CTRL Register (for Bus Off and Error interrupts) and in MCR for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

25 Analog-to-Digital Converter (ADC)

25.1 Overview

25.1.1 Device-specific features

- 1 ADC unit
- 10-bit resolution
- 27 input channels
  - 26 channels on 144-pin LQFP; 16 channels on 100-pin LQFP
  - Channel 15 dedicated to the internal 1.2 V rail
- Conversion time of 500 ns without sampling time
- Cross triggering unit (CTU)
- 4 analog watchdogs with interrupt capability for continuous hardware monitoring of as many as 4 analog input channels
- Clock stretching (with CTU pulse)
- Sampling and conversion time register CTR0 (internal precision channels)
- Left-aligned result format
- Right-aligned result format
- One Shot/Scan Modes
- Chain Injection Mode
- Power-down mode
- 2 different Abort functions allow aborting either single-channel conversion or chain conversion
- As many as 27 data registers for storing converted data. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- Auto-clock-off
- 2 modes of operation, each with DMA compatible interface
  - Normal Mode
  - CTU Control Mode
- External channels not supported

25.1.2 Device-specific pin configuration features

- For *Section 25.3.3, ADC sampling and conversion timing,"* f_{ck} = (1/2) PLL_CLK is true where the bit ADCLKSEL would be always 0 (default value), meaning that AD_clk is half of PLL_CLK. A clock prescaler (1 or 2) can be configured. The AD_clk has the same frequency of PLL_CLK or is half of PLL_CLK, depending on the value of the bit ADCLKSEL.
- CTUEN field in the MCR: Enables or disables CTU control mode
- Registers CDR[27..95] not used
- ADC channel 15 is internally connected to the core voltage

25.1.3 Device-specific implementation

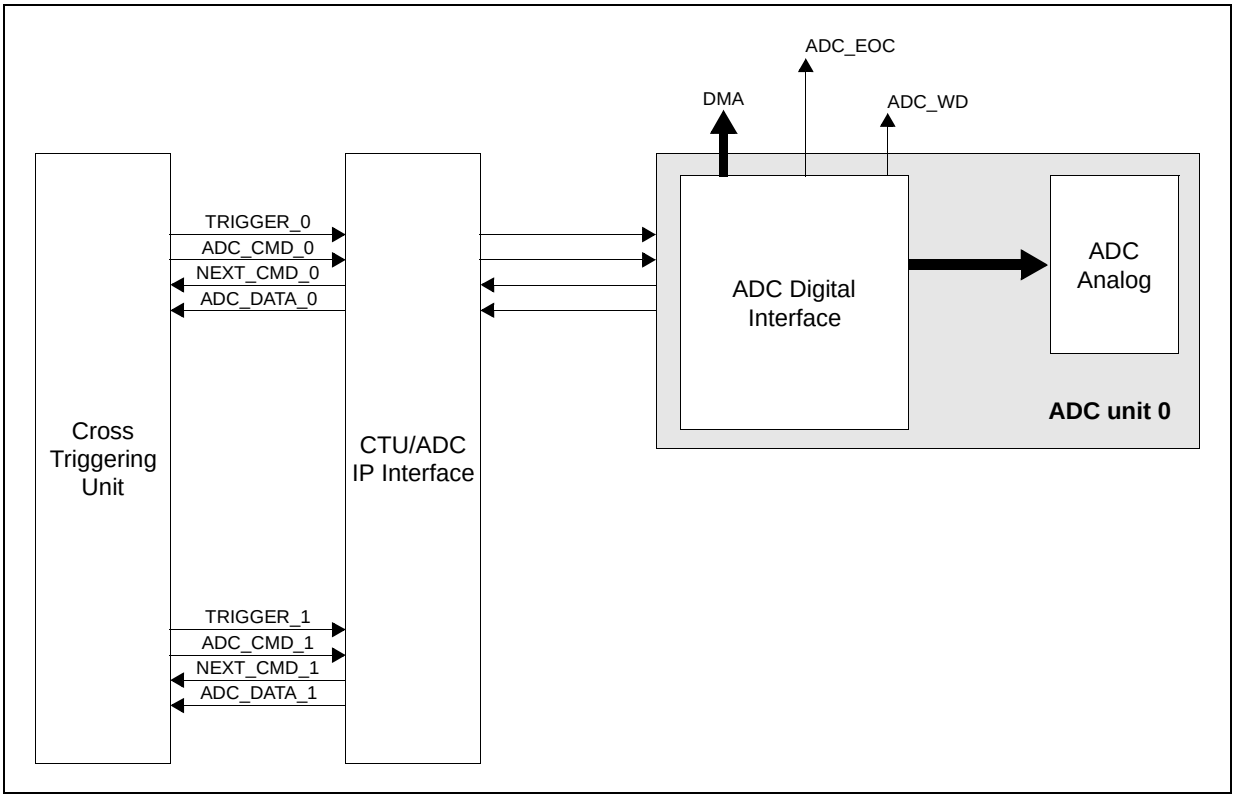


Figure 466. ADC implementation diagram

25.2 Introduction

The analog-to-digital converter (ADC) block provides accurate and fast conversions for a wide range of applications.

The ADC contains advanced features for normal or injected conversion. A conversion can be triggered by software or hardware (PIT).

The mask registers present within the ADC can be programmed to configure the channel to be converted.

A conversion timing register for configuring different sampling and conversion times is associated to each channel type.

Analog watchdogs allow continuous hardware monitoring.

25.3 Functional description

25.3.1 Analog channel conversion

Two conversion modes are available within the ADC:

- Normal conversion
- Injected conversion

Normal conversion

This is the normal conversion that the user programs by configuring the normal conversion mask registers (NCMR). Each channel can be individually enabled by setting '1' in the corresponding field of NCMR registers. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (NSTART bit in the Main Status Register (MSR) is reset).

Start of normal conversion

By programming the configuration bits in the Main Configuration Register (MCR), the normal conversion can be started in two ways:

- By software — The conversion chain starts when the MCR[NSTART] bit is set.
- By trigger — An on-chip internal signal triggers an ADC conversion. The settings in the MCR select how conversions are triggered based on these internal signals:
  - A rising/falling edge detected in the signal sets the MSR[NSTART] bit and starts the programmed conversion.
  - The conversion is started if and only if the MCR[NSTART] bit is set and the programmed level on the trigger signal is detected.

Table 450. Configurations for starting normal conversion

Type of conversion start	NSTART (in MCR)	NSTART (in MSR)	Result
Software	1	1	Conversion chain starts
Trigger	—	1	A falling or rising edge detected in a trigger signal sets the NSTART bit in the MSR and starts the programmed conversion.
	1	1	The conversion is started if the programmed level on the trigger signal is detected: the start of conversion is enabled if the external pin is low or high.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

If the content of all the normal conversion mask registers is zero (that is, no channel is selected) the conversion operation is considered completed and the interrupt ECH (see interrupt controller chapter for further details) is immediately issued after the start of conversion.

Normal conversion operating modes

Two operating modes are available for the normal conversion:

- One Shot
- Scan

To enter one of these modes, it is necessary to program the MCR[MODE] bit. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in *Figure 467*.

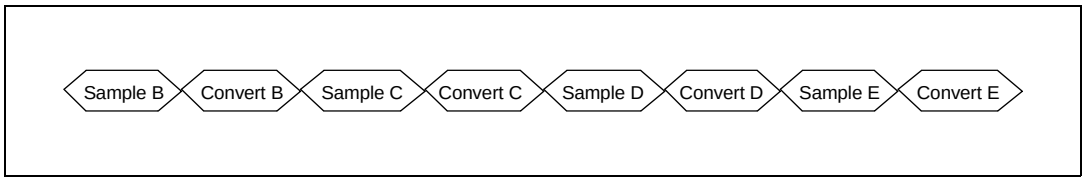


Figure 467. Normal conversion flow

In **One Shot Mode** (MODE = 0) a sequential conversion specified in the NCMR registers is performed only once. At the end of each conversion, the digital result of the conversion is stored in the corresponding data register.

Example 13 One Shot Mode (MODE = 0)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One Shot Mode. MODE = 0 is set for One Shot mode. Conversion starts from the channel B followed by conversion of channels D-E. At the end of conversion of channel E the scanning of channels stops.

The NSTART status bit in the MSR is automatically set when the Normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

In **Scan Mode** (MODE = 1), a sequential conversion of N channels specified in the NCMR registers is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored into the corresponding data register.

The MSR[NSTART] status bit is automatically set when the Normal conversion starts. Unlike One Shot Mode, the MCR[NSTART] bit is not reset. It can be reset by software when the user needs to stop scan mode. In that case, the ADC completes the current scan conversion and, after the last conversion, also resets the MSR[NSTART] bit.

Example 14 Scan Mode (MODE = 1)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the Scan Mode. MODE = 1 is set for Scan Mode. Conversion starts from the channel B followed by conversion of the channels D-E. At the end of conversion of channel E the scanning of channel B starts followed by conversion of the channels D-E. This sequence repeats itself till the MCR[NSTART] bit is cleared by software.

At the end of each conversion an End Of Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain interrupt is issued (if enabled by the corresponding mask bit in the IMR register).

Injected channel conversion

A conversion chain can be injected into the ongoing normal conversion by configuring the Injected Conversion Mask Registers (JCMR). As with normal conversion, each channel can be selected individually. This injected conversion (which can occur only in One Shot mode) interrupts the normal conversion (which can be in One Shot or Scan mode). When an injected conversion is inserted, ongoing normal channel conversion is aborted and the injected channel request is processed. After the last channel in the injected chain is converted, normal conversion resumes from the channel at which the normal conversion was aborted, as shown in *Figure 468*.

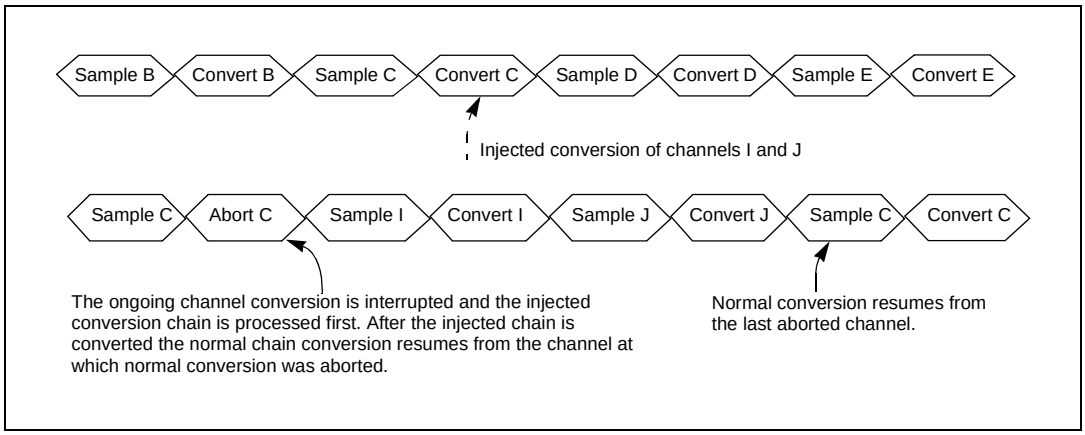


Figure 468. Injected sample/conversion sequence

The injected conversion can be started using two options:

- By software setting the MCR[JSTART]; the current conversion is suspended and the injected chain is converted. At the end of the chain, the JSTART bit in the MSR is reset and the normal chain conversion is resumed.
- By an internal trigger signal from the ETIMER when MCR[JTRGEN] is set; a programmed event (rising/falling edge depending on MCR[JEDGE]) on the signal coming from ETIMER starts the injected conversion by setting the MSR[JSTART]. At the end of the chain, the MSR[JSTART] is cleared and the normal conversion chain is resumed.

The MSR[JSTART] is automatically set when the Injected conversion starts. At the same time the MCR[JSTART] is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running injected conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the IMR[MSKJEOC]) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the IMR[MSKJEOC]).

If the content of all the injected conversion mask registers (JCMR) is zero (that is, no channel is selected) the JECH interrupt is immediately issued after the start of conversion.

Once started, injected chain conversion cannot be interrupted by any other conversion type (it can, however, be aborted; see *Section , Abort conversion*).

Abort conversion

Two different abort functions are provided.

- The user can abort the ongoing conversion by setting the MCR[ABORT] bit. The current conversion is aborted and the conversion of the next channel of the chain is immediately started. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is reset after the conversion of the next channel starts. The EOC interrupt corresponding to the aborted channel is not generated. This behavior is true for normal or Injected conversion modes. If the last channel of a chain is aborted, the end of chain is reported generating an ECH interrupt.
- It is also possible to abort the current chain conversion by setting the MCR[ABORTCHAIN] bit. In that case the behavior of the ADC depends on the MODE bit. If scan mode is disabled, the NSTART bit is automatically reset together with the MCR[ABORTCHAIN] bit. Otherwise, if the scan mode is enabled, a new chain conversion is started. The EOC interrupt of the current aborted conversion is not generated but an ECH interrupt is generated to signal the end of the chain.

  When a chain conversion abort is requested (ABORTCHAIN bit is set) while an injected conversion is running over a suspended Normal conversion, both injected chain and Normal conversion chain are aborted (both the NSTART and JSTART bits are also reset).

25.3.2 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL]. When this bit is set to ‘1’ the ADC clock has the same frequency as the PLL_CLK. Otherwise, the ADC clock is half of the PLL_CLK frequency. The ADCLKSEL bit can be written only in power-down mode.

When the internal divider is not enabled (ADCCLKSEL = 1), it is important that the associated clock divider in the clock generation module is ‘1’. This is needed to ensure 50% clock duty cycle.

The direct clock should basically be used only in low power mode when the device is using only the 16 MHz fast internal RC oscillator, but the conversion still requires a 16 MHz clock (an 8 MHz clock is not fast enough).

In all other cases, the ADC should use the clock divided by two internally.

Depending on the position of the rising edge of the signal internal trigger signal coming from the CTU, the ADC clock could also be stretched as illustrated in *Figure 469*.

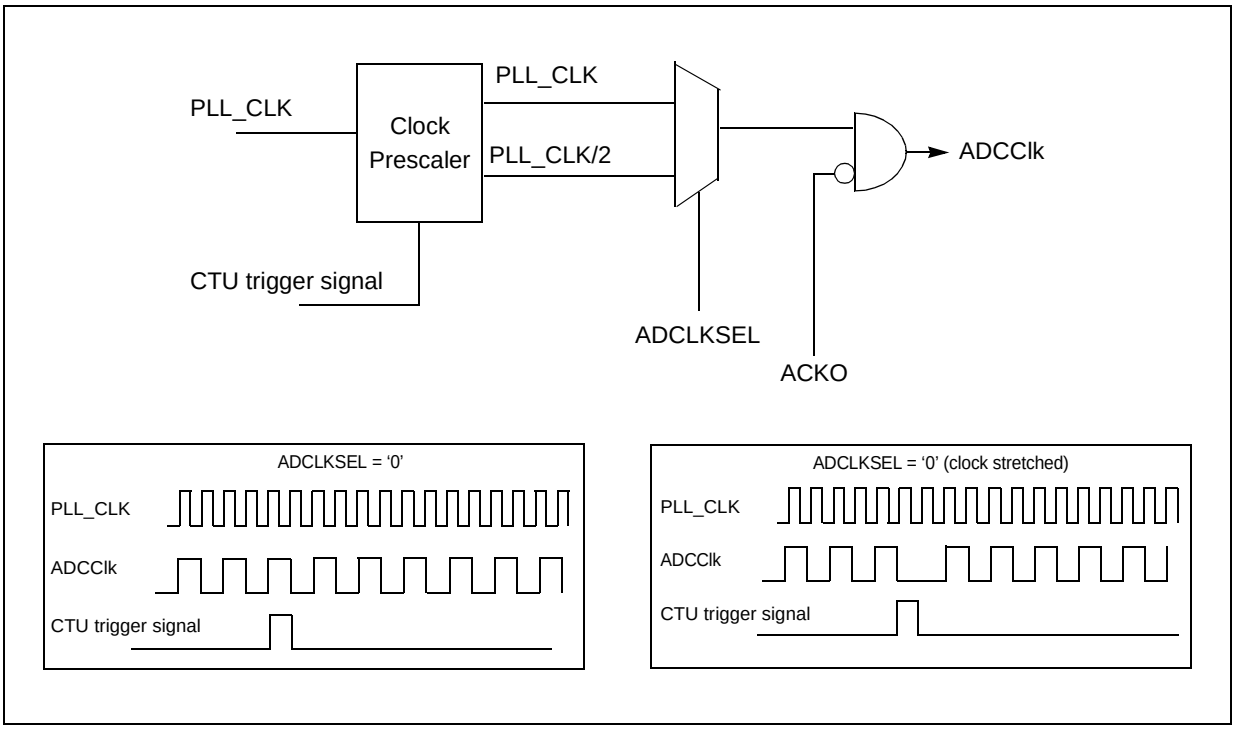


Figure 469. Prescaler simplified block diagram

The clock stretching is implemented if and only if ADCLKSEL = 0 (and clock is half of the PLL_CLK).

25.3.3 ADC sampling and conversion timing

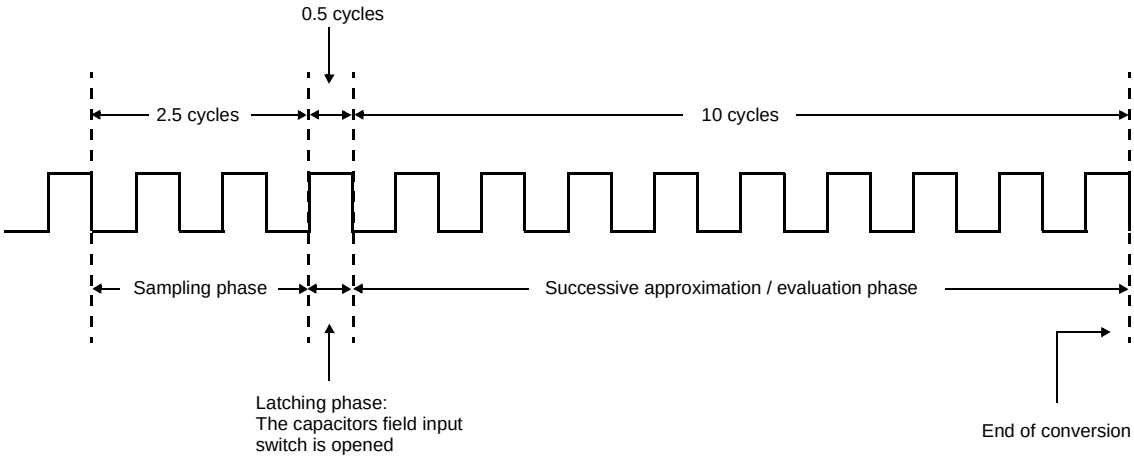
In order to support different loading and switching times, several different Conversion Timing registers (CTR) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

When a conversion is started, the ADC connects the internal sampling capacitor to the respective analog input pin, allowing the capacitance to charge up to the input voltage value. The time to load the capacitor is referred to as sampling time. After completion of the sampling phase, the evaluation phase starts and all the bits corresponding to the resolution of the ADC are estimated to provide the conversion result.

The conversion times are programmed via the bit fields of the CTR. Bit fields INPLATCH, INPCMP, and INPSAMP define the total conversion duration (T_{conv}) and in particular the partition between sampling phase duration (T_{sample}) and total evaluation phase duration (T_{eval}).

ADC_0

Figure 470 represents the sampling and conversion sequence.



Note: Operating conditions — INPLATCH = 0, INPSAMP = 3, INPCMP = 1 and Fadc clk = 20 MHz

Figure 470. Sampling and conversion timings

The sampling phase duration is:

$$T_{sample} = (INPSAMP - ndelay) \bullet T_{ck}$$

$$INPSAMP \geq 3$$

where ndelay is equal to 0.5 if INPSAMP is less than or equal to 06h, otherwise it is 1. INPSAMP must be greater than or equal to 3 (hardware requirement).

The total evaluation phase duration is:

$$T_{eval} = 10 \bullet T_{biteval} = 10 \bullet (INPCMP \bullet T_{ck})$$

$$(INPCMP \geq 1) \quad \text{and} \quad (INPLATCH < INPCMP)$$

INPCMP must be greater than or equal to 1 and INPLATCH must be less than INCMP (hardware requirements).

The total conversion duration is (not including external multiplexing):

$$T_{conv} = T_{sample} + T_{eval} + (ndelay \bullet T_{ck})$$

The timings refer to the unit T_{ck}, where f_{ck} = (1/2 x ADC peripheral set clock).

Table 451. ADC sampling and conversion timing at 5 V / 3.3 V for ADC0

Clock (MHz)	T_{ck} (µs)	INPSAMPLE(1)	Ndelay (2)	T_{sample} (3)	T_{sample}/T_{ck}	INPCMP	T_{eval} (µs)	INPLATCH	T_{conv} (µs)	T_{conv}/T_{ck}
6	0.167	4	0.5	0.583	3.500	1	1.667	0	2.333	14.000
7	0.143	4	0.5	0.500	3.500	1	1.429	0	2.000	14.000
8	0.125	5	0.5	0.563	4.500	1	1.250	0	1.875	15.000

Table 451. ADC sampling and conversion timing at 5 V / 3.3 V for ADC0 (continued)

Clock (MHz)	T_{ck} (µs)	INPSAMPLE(1)	Ndelay (2)	T_{sample} (3)	T_{sample}/T_{ck}	INPCMP	T_{eval} (µs)	INPLATCH	T_{conv} (µs)	T_{conv}/T_{ck}
16	0.063	9	1	0.500	8.000	1	0.625	0	1.188	19.000
32	0.031	17	1	0.500	16.000	2	0.625	1	1.156	37.000

1. Where: INPSAMPLE ≥ 3
2. Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1
3. Where: T_{sample} = (INPSAMP-N)T_{ck}; Must be ≥ 500 ns

Table 452. Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC0

INPCMP	INPLATCH	Max f_{ADC_clk}	Min f_{ADC_clk}
00/01	0	20+4%	6
	1	—	—
10	0	—	—
	1	32+4%	6
11	0	—	—
	1	32+4%	9

25.3.4 ADC CTU (Cross Triggering Unit)

Overview

The ADC cross triggering unit (CTU) is added to enhance the injected conversion capability of the ADC. The CTU contains multiple event inputs that can be used to select the channels to be converted from the appropriate event configuration register. The CTU generates a trigger output pulse of one clock cycle and outputs onto an internal data bus the channel to be converted. A single channel is converted for each request. After performing the conversion, the ADC returns the result on internal bus.

The conversion result is also saved in the corresponding data register and it is compared with watchdog thresholds if requested.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and the ADC are synchronous with the PLL_CLK in both cases.

CTU in control mode

In CTU control mode, the CPU is able to write in the ADC registers but it cannot start any conversion. Conversion requests can be generated only by the CTU trigger pulse. If a normal or injected conversion is requested, it is automatically discarded.

When a CTU trigger pulse is received with the injected channel number, the conversion starts. The CTUSTART bit is set automatically at this point and it is also automatically reset when CTU Control mode is disabled (CTUEN = ‘0’).

25.3.5 Presampling

Introduction

Presampling is used to precharge or discharge the ADC internal capacitor before it starts sampling of the analog input coming from the input pins. This is useful for resetting information regarding the last converted data or to have more accurate control of conversion speed. During presampling, the ADC samples the internally generated voltage.

Presampling can be enabled/disabled on a channel basis by setting the corresponding bits in the PSR registers.

After enabling the presampling for a channel, the normal sequence of operation will be Presampling + Sampling + Conversion for that channel. Sampling of the channel can be bypassed by setting the PRECONV bit in the PSCR. When sampling of a channel is bypassed, the sampled data of internal voltage in the presampling state is converted (*Figure 471*, *Figure 472*).

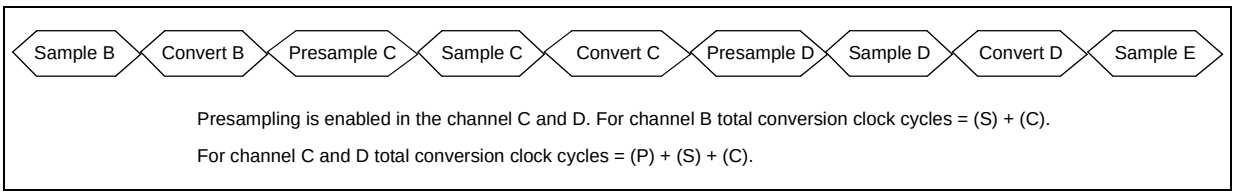


Figure 471. Presampling sequence

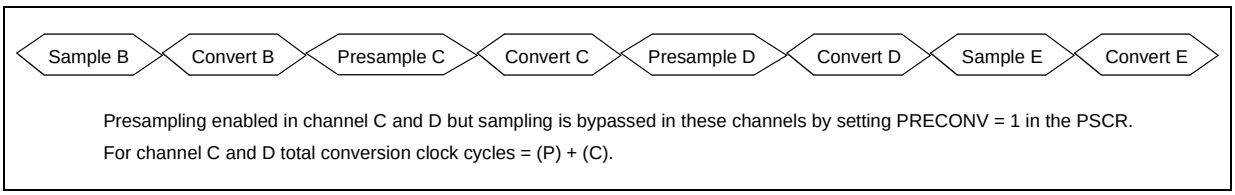


Figure 472. Presampling sequence with PRECONV = 1

Presampling channel enable signals

It is possible to select between two internally generated voltages V0 and V1 depending on the value of the PSCR[PREVAL] as shown in *Table 453*.

Table 453. Presampling voltage selection based on PREVALx fields

PSCR[PREVALx]	Presampling voltage
00	V0 = $V_{SS\_HV\_ADC}$
01	V1 = $V_{DD\_HV\_ADC}$
10	Reserved
11	Reserved

Three presampling value fields, one per channel type, in the PSCR make it possible to select different presampling values for each type.

25.3.6 Programmable analog watchdog

Introduction

The analog watchdogs are used for determining whether the result of a channel conversion lies within a given guarded area (as shown in *Figure 473*) specified by an upper and a lower threshold value named THRH and THRL respectively.

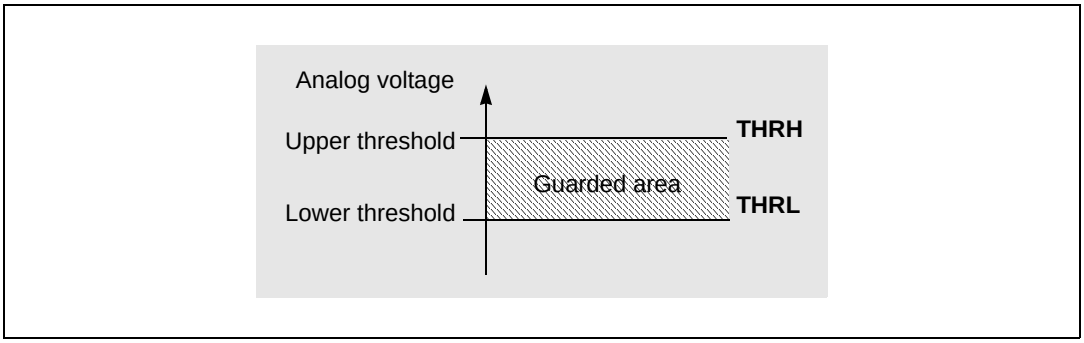


Figure 473. Guarded area

After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value lies outside that guarded area then corresponding threshold violation interrupts are generated. The comparison result is stored as WTISR[WDGxH] and WTISR[WDGxL] as explained in *Table 454*. Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

Table 454. Values of WDGxH and WDGxL fields

WDGxH	WDGxL	Converted data
1	0	converted data > THRH
0	1	converted data < THRL
0	0	THRL <= converted data <= THRH

The TRC[THRCH] field specifies the channel on which the analog watchdog is applied. The analog watchdog is enabled by setting the corresponding TRC[THREN].

The lower and higher threshold values for the analog watchdog are programmed using the THRHLR registers.

For example, if channel number 3 is to be monitored with threshold values in THRHLR1, then the TRC[THRCH] field is programmed to select channel number 3.

A set of threshold registers (THRHLRx and TRCx) can be linked only to a single channel for a particular THRCH value. If another channel is to be monitored with same threshold values, then the TRCx[THRCH] must be programmed again.

Note: If the higher threshold for the analog watchdog is programmed lower than the lower threshold and the converted value is less than the lower threshold, then the WDGxL interrupt for the low threshold violation is set, else if the converted value is greater than the lower threshold (consequently also greater than the higher threshold) then the interrupt

WDGxH for high threshold violation is set. Thus, the user should avoid that situation as it could lead to misinterpretation of the watchdog interrupts.

Analog watchdog functionality

For each input channel the result of the comparison is reflected in the THROP bit in TRC register based on the converted analog values received by the analog watchdogs:

- If the converted data value is lower than the lower threshold then the THROP bit in TRC register will be set to 1.
- If the converted voltage is higher than the higher threshold then the THROP bit in TRC register will be set to 0.
- If the converted voltage lies between the upper and the lower threshold guard window then THROP bit in TRC register will keep its logic value.

The logic level of the THROP bit can be programmed by software. In fact, the user can decide to keep the behavior described or to invert the output logic level by setting the THRINV bit in the TRC register.

An example of the operation is shown in *Table 455*.

Table 455. Example for Analog watchdog operation

Converted data watchdog[x]	Upper threshold watchdog[x]	Lower threshold watchdog[x]	THRINV watchdog[x]	THROP[x]
155h	055h	000h	0	0
055h	1ffh	088h	0	1
155h	055h	000h	1	1
055h	1ffh	088h	1	0

25.3.7 DMA functionality

A DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR registers. The DMAR masking registers must be programmed before starting any conversion. There is one DMAR per channel type.

The DMA transfers can be enabled using the DMAEN bit of DMAE register. When the DCLR bit of DMAE register is set, the DMA request is cleared the register enabled for DMA transfer has been read.

25.3.8 Interrupts

The ADC generates the following maskable interrupt signals:

- EOC (end of conversion) interrupt request
- ECH (end of chain) interrupt request
- JEOC (end of injected conversion) interrupt request
- JECH (end of injected chain) interrupt request
- EOCTU (end of CTU conversion) interrupt request
- WDGxL and WDGxH (watchdog threshold) interrupt requests

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in register description for ISR and IMR.

The analog watchdog interrupts are handled by two registers WTISR (Watchdog Threshold Interrupt Status Register) and WTIMR (Watchdog Threshold Interrupt Mask Register) in order to check and enable the interrupt request to the INTC module. The Watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored.

25.3.9 Power-down mode

The analog part of the ADC can be put in low power mode by setting the MCR[PWDN]. After releasing the reset signal the ADC analog module is kept in power-down mode by default, so this state must be exited before starting any operation by resetting the appropriate bit in the MCR.

The power-down mode can be requested at any time by setting the MCR[PWDN]. If a conversion is ongoing, the ADC must complete the conversion before entering the power down mode. In fact, the ADC enters power-down mode only after completing the ongoing conversion. Otherwise, the ongoing operation should be aborted manually by resetting the NSTART bit and using the ABORTCHAIN bit.

MSR[ADCSTATUS] bit is set only when ADC enters power-down mode.

After the power-down phase is completed the process ongoing before the power-down phase must be restarted manually by setting the appropriate MCR[START] bit.

Resetting MCR[PWDN] bit and setting MCR[NSTART] or MCR[JSTART] bit during the same cycle is forbidden.

If a CTU trigger pulse is received during power-down, it is discarded.

If the CTU is enabled and the MSR[CTUSTART] bit is ‘1’, then the MCR[PWDN] bit cannot be set.

25.3.10 Auto-clock-off mode

To reduce power consumption during the IDLE mode of operation (without going into power-down mode), an “auto-clock-off” feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

Note: The auto-clock-off feature cannot operate when the digital interface runs at the same rate as the analog interface. This means that when MCR.ADCCLKSEL = 1, the analog clock will not shut down in IDLE mode.

25.4 Register descriptions

25.4.1 Introduction

Table 456 lists ADC registers with their address offsets and reset values.

Table 456. ADC digital registers

Offset from base address 0xFFE0_0000	Register name	Location
0x0000	Main Configuration Register (MCR)	*on page 824*
0x0004	Main Status Register (MSR)	*on page 826*
0x0008–0x000F	Reserved	
0x0010	Interrupt Status Register (ISR)	*on page 828*
0x0014–0x001F	Reserved	
0x0020	Interrupt Mask Register (IMR)	*on page 828*
0x0024–0x002F	Reserved	
0x0030	Watchdog Threshold Interrupt Status Register (WTISR)	*on page 830*
0x0034	Watchdog Threshold Interrupt Mask Register (WTIMR)	*on page 830*
0x0038–0x003F	Reserved	
0x0040	DMA Enable Register (DMAE)	*on page 832*
0x0044	DMA Channel Select Register 0 (DMAR0)	*on page 833*
0x0048–0x004F	Reserved	
0x0050	Threshold Control Register 0 (TRC0)	*on page 834*
0x0054	Threshold Control Register 1 (TRC1)	*on page 834*
0x0058	Threshold Control Register 2 (TRC2)	*on page 834*
0x005C	Threshold Control Register 3 (TRC3)	*on page 834*
0x0060	Threshold Register 0 (THRHLR0)	*on page 835*
0x0064	Threshold Register 1 (THRHLR1)	*on page 835*
0x0068	Threshold Register 2 (THRHLR2)	*on page 835*
0x006C	Threshold Register 3 (THRHLR3)	*on page 835*
0x0070–0x007F	Reserved	
0x0080	Presampling Control Register (PSCR)	*on page 835*
0x0084	Presampling Register 0 (PSR0)	*on page 835*
0x0088–0x0093	Reserved	
0x0094	Conversion Timing Register 0 (CTR0)	*on page 837*
0x0098–0x00A3	Reserved	
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	*on page 837*
0x00A8–0x00B3	Reserved	
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	*on page 838*
0x00B8–00C7	Reserved	
0x00C8	Power-down Exit Delay Register (PDEDR)	*on page 839*
0x00CC–0x00FF	Reserved	

Table 456. ADC digital registers

Offset from base address 0xFFE0_0000	Register name	Location
0x0100	Channel 0 Data Register (CDR0)	*on page 839*
0x0104	Channel 1 Data Register (CDR1)	*on page 839*
0x0108	Channel 2 Data Register (CDR2)	*on page 839*
0x010C	Channel 3 Data Register (CDR3)	*on page 839*
0x0110	Channel 4 Data Register (CDR4)	*on page 839*
0x0114	Channel 5 Data Register (CDR5)	*on page 839*
0x0118	Channel 6 Data Register (CDR6)	*on page 839*
0x011C	Channel 7 Data Register (CDR7)	*on page 839*
0x0120	Channel 8 Data Register (CDR8)	*on page 839*
0x0124	Channel 9 Data Register (CDR9)	*on page 839*
0x0128	Channel 10 Data Register (CDR10)	*on page 839*
0x012C	Channel 11 Data Register (CDR11)	*on page 839*
0x0130	Channel 12 Data Register (CDR12)	*on page 839*
0x0134	Channel 13 Data Register (CDR13)	*on page 839*
0x0138	Channel 14 Data Register (CDR14)	*on page 839*
0x013C	Channel 15 Data Register (CDR15)	*on page 839*
0x0140	Channel 16 Data Register (CDR16)	*on page 839*
0x0144	Channel 17 Data Register (CDR17)	*on page 839*
0x0148	Channel 18 Data Register (CDR18)	*on page 839*
0x014C	Channel 19 Data Register (CDR19)	*on page 839*
0x0150	Channel 20 Data Register (CDR20)	*on page 839*
0x0154	Channel 21 Data Register (CDR21)	*on page 839*
0x0158	Channel 22 Data Register (CDR22)	*on page 839*
0x015C	Channel 23 Data Register (CDR23)	*on page 839*
0x0160	Channel 24 Data Register (CDR24)	*on page 839*
0x0164	Channel 25 Data Register (CDR25)	*on page 839*
0x0168	Channel 26 Data Register (CDR26)	*on page 839*

25.4.2 Control logic registers

Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Figure 474. Main Configuration Register (MCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OWREN	WLSIDE	MODE	0	0	0	0	NSTART	0	JTRGEN	JEDGE	JSTART	0	0	CTUEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	ADCLK SEL	ABORT CHAIN	ABORT	ACKO	0	0	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 457. MCR field descriptions

Field	Description
OWREN	Overwrite enable This bit enables or disables the functionality to overwrite unread converted data. 0 Prevents overwrite of unread converted data; new result is discarded 1 Enables converted data to be overwritten by a new conversion
WLSIDE	Write left/right-aligned 0 The conversion data is written right-aligned. 1 Data is left-aligned (from 15 to (15 – resolution + 1)). The WLSIDE bit affects all the CDR registers simultaneously. See *Figure 490* and *Figure 490*.
MODE	One Shot/Scan 0 One Shot Mode—Configures the normal conversion of one chain. 1 Scan Mode—Configures continuous chain conversion mode; when the programmed chain conversion is finished it restarts immediately.
NSTART	Normal Start conversion Setting this bit starts the chain or scan conversion. Resetting this bit during scan mode causes the current chain conversion to finish, then stops the operation. This bit stays high while the conversion is ongoing (or pending during injection mode). 0 Causes the current chain conversion to finish and stops the operation 1 Starts the chain or scan conversion
JTRGEN	Injection external trigger enable 0 External trigger disabled for channel injection 1 External trigger enabled for channel injection
JEDGE	Injection trigger edge selection Edge selection for external trigger, if JTRGEN = 1. 0 Selects falling edge for the external trigger 1 Selects rising edge for the external trigger
JSTART	Injection start Setting this bit will start the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.

Table 457. MCR field descriptions (continued)

Field	Description
CTUEN	Cross trigger unit conversion enable 0 CTU triggered conversion disabled 1 CTU triggered conversion enabled
ADCLKSEL	Analog clock select This bit can only be written when ADC in Power-Down mode 0 ADC clock frequency is half Peripheral Set Clock frequency 1 ADC clock frequency is equal to Peripheral Set Clock frequency
ABORTCHAIN	Abort Chain When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested. 0 Conversion is not affected 1 Aborts the ongoing chain conversion
ABORT	Abort Conversion When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked. If it is set during a scan chain, only the ongoing conversion is aborted and the next conversion is performed as planned. 0 Conversion is not affected 1 Aborts the ongoing conversion
ACKO	Auto-clock-off enable If set, this bit enables the Auto clock off feature. 0 Auto clock off disabled 1 Auto clock off enabled
PWDN	Power-down enable When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode. 0 ADC is in normal mode 1 ADC has been requested to power down

Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Figure 475. Main Status Register (MSR)

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	NSTART	JABORT	0	0	JSTART	0	0	0	CTUSTART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHADDR							0	0	0	ACK0	0	0	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 458. MSR field descriptions

Field	Description
NSTART	This status bit is used to signal that a Normal conversion is ongoing.
JABORT	This status bit is used to signal that an Injected conversion has been aborted. This bit is reset when a new injected conversion starts.
JSTART	This status bit is used to signal that an Injected conversion is ongoing.
CTUSTART	This status bit is used to signal that a CTU conversion is ongoing.
CHADDR	Current conversion channel address This status field indicates current conversion channel address.
ACKO	Auto-clock-off enable This status bit is used to signal if the Auto-clock-off feature is on.
ADCSTATUS	The value of this parameter depends on ADC status: 000 IDLE — The ADC is powered up but idle. 001 Power-down — The ADC is powered down. 010 Wait state — The ADC is waiting for an external multiplexer. This occurs only when the DSDR register is non-zero. 011 Reserved 100 Sample — The ADC is sampling the analog signal. 101 Reserved 110 Conversion — The ADC is converting the sampled signal. 111 Reserved

Note: MSR[JSTART] is automatically set when the injected conversion starts. At the same time MCR[JSTART] is reset, allowing the software to program a new start of conversion.

The JCMR registers do not change their values.

25.4.3 Interrupt registers

Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Figure 476. Interrupt Status Register (ISR)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	EOCTU	JEOC	JECH	EOC	ECH
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 459. ISR field descriptions

Field	Description
EOCTU	End of CTU Conversion interrupt flag When this bit is set, an EOCTU interrupt has occurred.
JEOC	End of Injected Channel Conversion interrupt flag When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt flag When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt flag When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt flag When this bit is set, an ECH interrupt has occurred.

Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Figure 477. Interrupt Mask Register (IMR)

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	MSKEOCTU	MSKJEOC	MSKJECH	MSKEOC	MSKECH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 460. IMR field descriptions

Field	Description
MSKEOCTU	Mask for end of CTU conversion (EOCTU) interrupt When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask for end of injected channel conversion (JEOC) interrupt When set, the JEOC interrupt is enabled.
MSKJECH	Mask for end of injected chain conversion (JECH) interrupt When set, the JECH interrupt is enabled.
MSKEOC	Mask for end of channel conversion (EOC) interrupt When set, the EOC interrupt is enabled.
MSKECH	Mask for end of chain conversion (ECH) interrupt When set, the ECH interrupt is enabled.

Watchdog Threshold Interrupt Status Register (WTISR)

Figure 478. Watchdog Threshold Interrupt Status Register (WTISR)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	WDG3H	WDG2H	WDG1H	WDG0H	WDG3L	WDG2L	WDG1L	WDG0L
W									w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 461. WTISR field descriptions

Field	Description
WDGxH	This corresponds to the status flag generated on the converted value being higher than the programmed higher threshold (for [x = 0..3]).
WDGxL	This corresponds to the status flag generated on the converted value being lower than the programmed lower threshold (for [x = 0..3]).

Watchdog Threshold Interrupt Mask Register (WTIMR)

Figure 479. Watchdog Threshold Interrupt Mask Register (WTIMR)

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MSKWDG3H	MSKWDG2H	MSKWDG1H	MSKWDG0H	MSKWDG3L	MSKWDG2L	MSKWDG1L	MSKWDG0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 462. WTIMR field descriptions

Field	Description
MSKWDGxH	This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0..3]). When set the interrupt is enabled.
MSKWDGxL	This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0..3]). When set the interrupt is enabled.

25.4.4 DMA registers

DMA Enable (DMAE) register

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

Figure 480. DMA Enable (DMAE) register

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DCLR	DMAEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 463. DMAE field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request cleared by Acknowledge from DMA controller 1 DMA request cleared on read of data registers
DMAEN	DMA global enable 0 DMA feature disabled 1 DMA feature enabled

DMA Channel Select Register (DMAR[0])

DMAR0 = Enable bits for channel 0 to 26 (precision channels)

Figure 481. DMA Channel Select Register 0 (DMAR0)

Address: Base + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DMA 26	DMA 25	DMA 24	DMA 23	DMA 22	DMA 21	DMA 20	DMA 19	DMA 18	DMA 17	DMA 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA 15	DMA 14	DMA 13	DMA 12	DMA 11	DMA 10	DMA 9	DMA 8	DMA 7	DMA 6	DMA 5	DMA 4	DMA 3	DMA 2	DMA 1	DMA 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 464. DMARx field descriptions

Field	Description
DMAn	DMA enable When set (DMAn = 1), channel n is enabled to transfer data in DMA mode.

25.4.5 Threshold registers

Introduction

The Threshold registers store the user programmable lower and upper thresholds' values. The inverter bit and the mask bit for mask the interrupt are stored in the TRC registers.

Threshold Control Register (TRC*x*, *x* = [0..3])

Figure 482. Threshold Control Register (TRC*x*, *x* = [0..3])

Address:
Base + 0x0050 (TRC0)
Base + 0x0054 (TRC1)
Base + 0x0058 (TRC2)
Base + 0x005C (TRC3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	THREN	THRINV	THROP	0	0	0	0	0	0	THRCH						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 465. TRC*x* field descriptions

Field	Description
THREN	Threshold enable When set, this bit enables the threshold detection feature for the selected channel.
THRINV	Invert the output pin Setting this bit inverts the behavior of the threshold output pin.
THROP	This bit reflects the output pin status.
THRCH	Choose the channel for threshold comparison.

Threshold Register (THRHLR[0:3])

The four THRHLR*n* registers store the user-programmable thresholds' 10-bit values.

Figure 483. Threshold Register (THRHLR[0:3])

Address: Base + 0x0060 (THRHLR0)
Base + 0x0064 (THRHLR1)
Base + 0x0068 (THRHLR2)
Base + 0x006C (THRHLR3)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	THRH									
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	THRL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 466. THRHLR*x* field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

25.4.6 Presampling registers

Presampling Control Register (PSCR)

Figure 484. Presampling Control Register (PSCR)

Address: Base + 0x0080
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	PREVAL0		PRE CONV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 467. PSCR field descriptions

Field	Description
PREVAL0	Internal voltage selection for presampling 00 Select the V0 internal presampling voltage 01 Select the V1 internal presampling voltage 10 Reserved 11 Reserved
PRECONV	Convert presampled value If bit PRECONV is set, presampling is followed by the conversion. Sampling will be bypassed and conversion of presampled data will be done.

Presampling Register (PSR[0])

PSR0 = Enable bits of presampling for channel 0 to 26 (precision channels)

Figure 485. Presampling Register 0 (PSR0)

Address: Base + 0x0084 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PRES 26	PRES 25	PRES 24	PRES 23	PRES 22	PRES 21	PRES 20	PRES 19	PRES 18	PRES 17	PRES 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRES 15	PRES 14	PRES 13	PRES 12	PRES 11	PRES 10	PRES 9	PRES 8	PRES 7	PRES 6	PRES 5	PRES 4	PRES 3	PRES 2	PRES 1	PRES 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 468. PSR field descriptions

Field	Description
PRESn	Presampling enable When set (PRESn = 1), presampling is enabled for channel n.

25.4.7 Conversion Timing Registers CTR[0]

CTR0 = associated to internal precision channels (from 0 to 26)

Figure 486. Conversion Timing Registers CTR[0]

Address: Base + 0x0094 (CTR0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPLATCH	0	OFFSHIFT		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1

Table 469. CTR field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration
OFFSHIFT	Configuration for offset shift characteristic 00 No shift (that is the transition between codes 000h and 001h) is reached when the A_{VIN} (analog input voltage) is equal to 1 LSB. 01 Transition between code 000h and 001h is reached when the A_{VIN} is equal to1/2 LSB 10 Transition between code 00h and 001h is reached when the A_{VIN} is equal to 0 11 Not used Available only on CTR0
INPCMP	Configuration bits for comparison phase duration
INPSAMP	Configuration bits for sampling phase duration

25.4.8 Mask registers

Introduction

The Mask registers are used to program the 27 input channels that are converted during Normal and Injected conversion.

Normal Conversion Mask Registers (NCMR[0])

NCMR0 = Enable bits of normal sampling for channel 0 to 26 (precision channels)

Figure 487. Normal Conversion Mask Register 0 (NCMR0)

Address: Base + 0x00A4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	CH26	CH25	CH24	CH23	CH22	CH21	CH20	CH19	CH18	CH17	CH16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 470. NCMR field descriptions

Field	Description
CHn	Sampling enable When set Sampling is enabled for channel n.

Injected Conversion Mask Registers (JCMR[0])

JCMR0 = Enable bits of injected sampling for channel 0 to 26 (precision channels)

Figure 488. Injected Conversion Mask Register 0 (JCMR0)

Address: Base + 0x00B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	CH26	CH25	CH24	CH23	CH22	CH21	CH20	CH19	CH18	CH17	CH16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 471. JCMR field descriptions

Field	Description
CHn	Sampling enable When set, sampling is enabled for channel n.

25.4.9 Delay registers

Power-Down Exit Delay Register (PDEDR)

Figure 489. Power-Down Exit Delay Register (PDEDR)

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PDED							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 472. PDEDR field descriptions

Field	Description
PDED	Delay between the power-down bit reset and the start of conversion. The delay is to allow time for the ADC power supply to settle before commencing conversions. The power down delay is calculated as: PDED x 1/frequency of ADC clock.

25.4.10 Data registers

Introduction

ADC conversion results are stored in data registers. There is one register per channel.

Channel Data Registers (CDR[0..26])

CDR[0..26] = precision channels

Each data register also gives information regarding the corresponding result as described below.

Figure 490. Channel Data Registers (CDR[0..26])

Address: See *Table 456* Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA[0:9] (MCR[WLSIDE] = 0)									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDATA[0:9] (MCR[WLSIDE] = 1)										0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 473. CDR field descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]: – When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read. – When OWREN = 1, then OVERW flags the CDATA field overwrite status. 0 Converted data has not been overwritten 1 Previous converted data has been overwritten before having been read
RESULT	This bit reflects the mode of conversion for the corresponding channel. 00 Data is a result of Normal conversion mode 01 Data is a result of Injected conversion mode 10 Data is a result of CTU conversion mode 11 Reserved
CDATA	Channel 0- converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this field can be changed as shown in *Figure 490*.

26 Cross Triggering Unit (CTU)

26.1 Introduction

The cross triggering unit (CTU) is intended to completely avoid CPU involvement in the time acquisitions of state variables during the control cycle that can be the PWM cycle, the half PWM cycle or a number of PWM cycles. In such cases the pre-setting of the acquisition times needs to be completed during the previous control cycle, where the actual acquisitions are to be made, and a double-buffered structure for the CTU registers is used, in order to activate the new settings at the beginning of the next control cycle. Additionally, four FIFOs inside the CTU are available to store the ADC results.

26.2 CTU overview

The CTU receives various incoming signals from different sources (timers, position decoder and/or external pins). These signals are then processed to generate as many as eight trigger events. An input can be a rising edge, a falling edge or both, edges of each incoming signal. The output can be a pulse or a command (or a stream of consecutive commands for over-sampling support) or both, to one or more peripherals (for example, ADC or timers).

The CTU interfaces to the following peripherals:

- Timers—2 inputs
- GPIO—1 external input signal

The 16 input signals are digital signals and the CTU must be able to detect a rising and/or a falling edge for each of them.

The CTU comprises the following:

- Input signals interface
- User interface (such as configuration registers)
- ADC interface
- Timers interface

The block diagram of the CTU is shown in *Figure 491*.

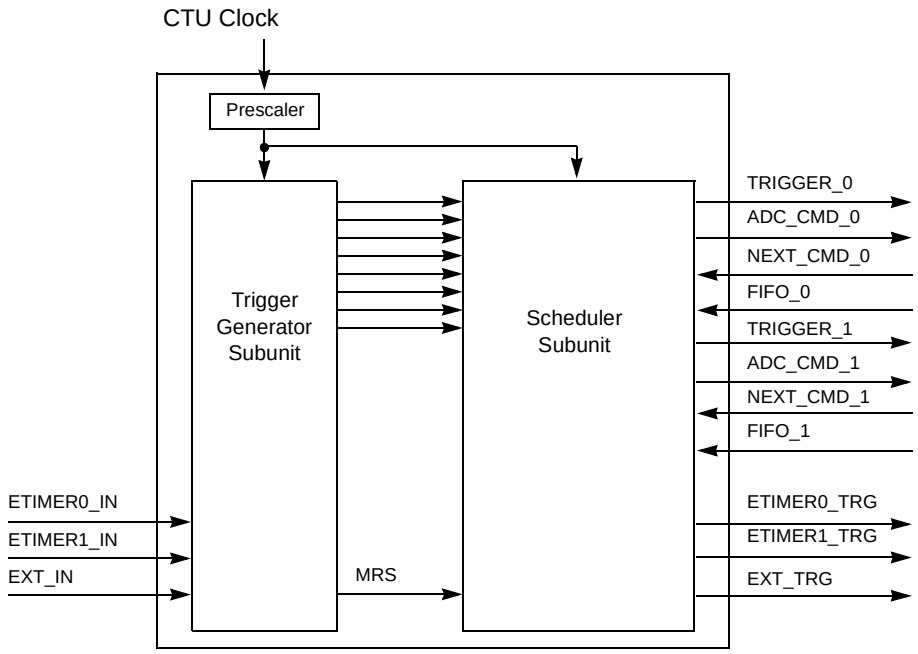


Figure 491. Cross triggering unit diagram

The CTU consists of two subunits:

- Trigger generator
- Scheduler

The trigger generator subunit handles incoming signals, selecting for each signal, the active edges to generate the Master Reload signal, and generates as many as eight trigger events (signals). The scheduler subunit generates the trigger event output according to the occurred trigger event (signal).

26.3 Functional description

The following describes the functionality of the CTU.

26.3.1 Trigger events features

The TGS has the capability to generate as many as eight trigger events. Each trigger event has the following characteristics:

- Generation of the trigger event is sequential in time
- The triggers list uses eight 16-bit double-buffered registers
- On each Master Reload Signal (MRS), the new triggers list is loaded
- The triggers list is reloaded on a MRS occurrence, only if the reload enable bit is set

26.3.2 Trigger generator subunit (TGS)

The trigger generator subunit has the following two modes:

- Triggered mode—Each event source for the incoming signals can generate as many as eight trigger event outputs. For the ADC, a commands list is entered by the CPU, and each event source can generate as many as eight commands or streams of commands.
- Sequential mode—Each event source for the incoming signals can generate one trigger event output, the next event source generates the next trigger event output, and so on in a predefined sequence. For the ADC, a commands list is entered by the CPU and the sequence of the selected incoming trigger events generate commands or stream of commands.

The TGS Mode is selected using the TGS_M bit in the TGS Control Register.

26.3.3 TGS in triggered mode

The structure of the TGS in Triggered mode is shown in *Figure 492*.

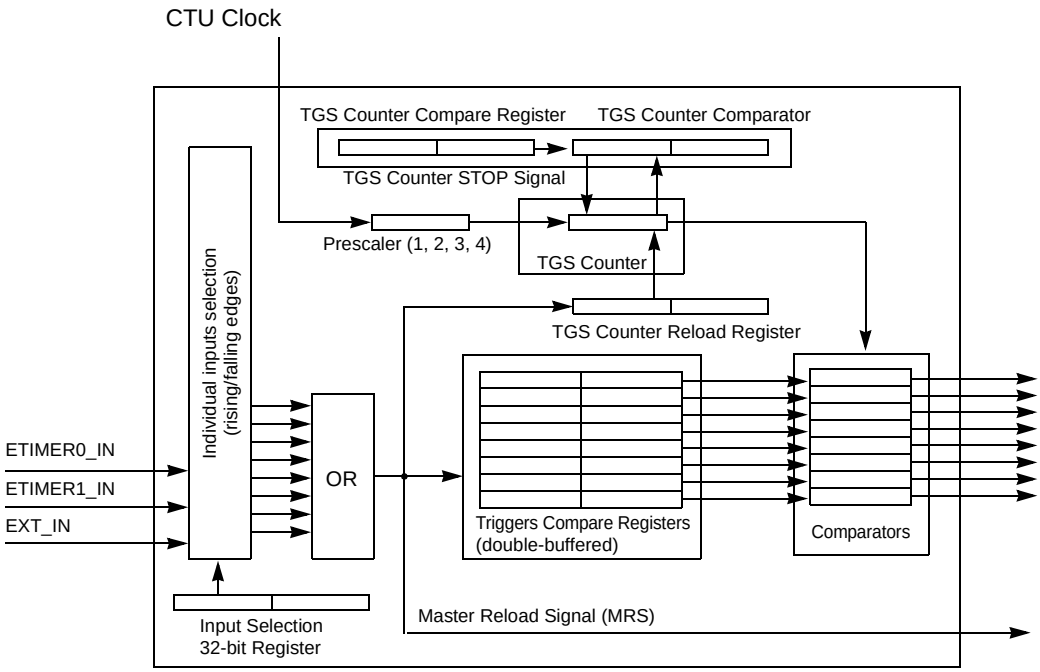


Figure 492. TGS in triggered mode

The TGS has 16 input signals, each of which is selected from the input selection register (TGSISR), selecting the states inactive, rising, falling or both. Depending on the selection, as many as 32 input events can be enabled. These signals are ORed in order to generate the MRS. The MRS, at the beginning of the control cycle *n* (defined by the MRS occurrence), preloads the TGS counter register, using the preload value written into the

double-buffered register (TGSCRR), during the control cycle *n* – 1 and reloads all the double-buffered registers (such as Trigger Compare registers, TGSCR, TGSCRR itself).

The triggers list registers consist of eight compare registers. Each triggers list register is associated with a comparator. On reload (MRS occurrence), the comparators are disabled. One TGS clock cycle is necessary to enable them and to start the counting. The MRS is output together with individual trigger signals. The MRS can be performed by hardware or by software. The MRS_SG bit in the CTU control register, if set to 1, generates equivalent software MRS (that is, resets/reloads TGS Counter and reloads all double-buffered registers). This bit is cleared by each hardware or software MRS occurrence.

The TGS counter compare register and the TGS counter comparator are used to stop the TGS counter when it reaches the value stored in the TGS counter compare register before an MRS occurs.

The prescaler for TGS and SU can be 1,2,3,4 (PRES bits in the TGS Control Register).

An example timing for the TGS in Triggered Mode is shown in *Figure 493*. The red arrows indicate the MRS occurrences, while the black arrows indicate the trigger event occurrences, with the relevant delay in respect to the last MRS occurrence.

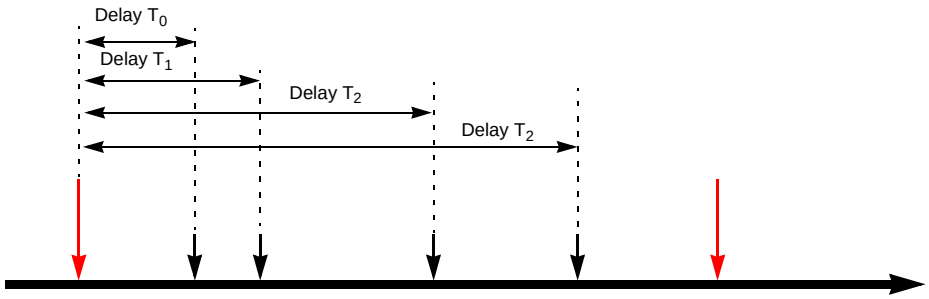


Figure 493. Example timing for TGS in triggered mode

26.3.4 TGS in sequential mode

The structure of the TGS in sequential mode is shown in *Figure 494*.

The 32 input events (16 signals with two edges for signal), which can be individually enabled, are ORed in order to generate the event signal (ES). The ES enables the reload of the TGS counter register and pilots the 3-bit counter in order to select the next active trigger. One of the 32 input events can be selected, through the MRS_SM (master reload selection sequential mode) bits in the TGS control register, to be the MRS, that enables the reload of the triggers list and resets the 3-bit counter (incoming events counter), that is, the MRS is the signal linked with the control cycle defined as the time window between two consecutive MRSs. In this mode, each incoming event sequentially enables only one trigger event through the 3-bit counter and the MUX. The MUX is a selection switch that enables, according to the number of event signals occurred, only one of the eight trigger signals to the scheduler subunit. Sequences of as many as eight trigger events can be supported within this control cycle.

For the other features see the previous paragraphs.

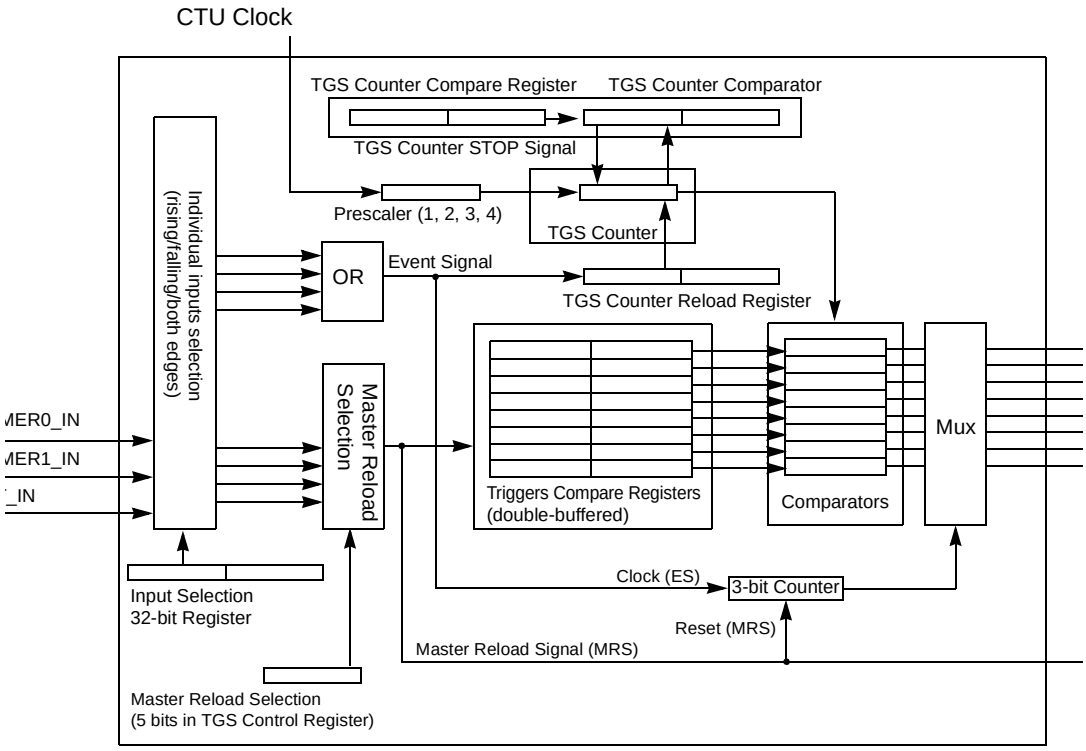


Figure 494. TGS in sequential mode

An example timing diagram for TGS in sequential mode is shown in *Figure 495*. The red arrows indicate the MRS occurrences and ES occurrences, while the black arrows indicate the trigger event occurrences with the relevant delay in respect to the ES occurrence. The first red arrow indicates the first ES occurrence, which is also the MRS.

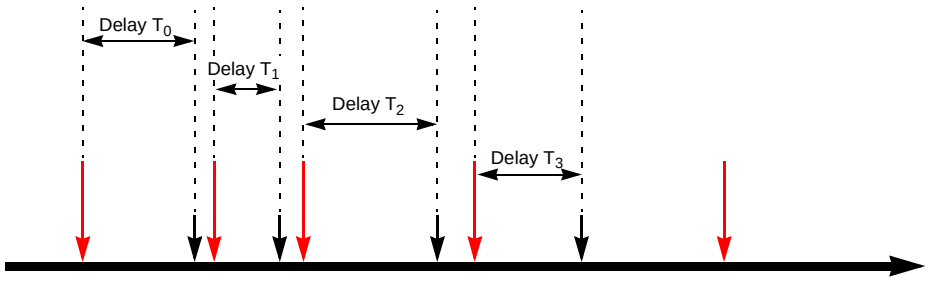


Figure 495. Example timing for TGS in sequential mode

26.3.5 TGS counter

The TGS counter is able to count from negative to positive, that is, from 0x8000 to 0x7FFF. *Figure 496* shows examples in order to explain the TGS counter counts. The compare operation to stop the TGS counter is not enabled during the first counting cycle, in order to allow the counting, if the value of the TGSCRR is the same as the value of the TGSCCR.

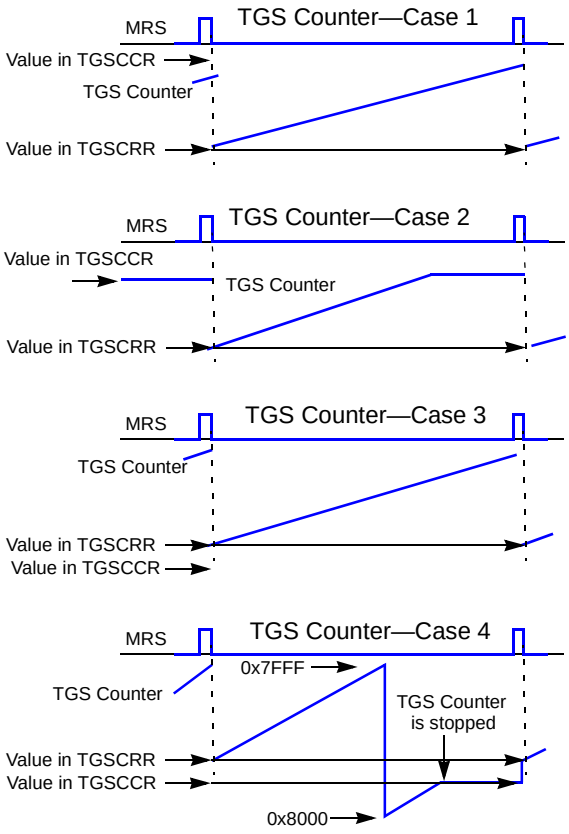


Figure 496. TGS counter cases

26.4 Scheduler subunit (SU)

The structure of the SU is shown in *Figure 497*.

The SU generates the trigger event output according to the occurred trigger event, and it has the same functionality in both TGS modes (triggered mode and sequential mode). Each of the four SU outputs:

- ADC command or ADC stream of commands
- eTimer1 pulse (ETIMER0_TRG internally connected to eTimer_0 AUX0)
- eTimer2 pulse (ETIMER1_TRG internally connected to eTimer_1 AUX0
- External trigger pulse

can be linked to any of eight trigger events by the Trigger Handler block. Each trigger event can be linked to one or more SU outputs.

If two events at the same time are linked to the same output only one output is generated and an error is provided. The output is generated using the trigger with the lowest index. For example, if trigger 0 and trigger 1 are linked to the ADC output and they occur together, an error is generated and the output linked with the trigger 0 is generated.

When a trigger is linked to the ADC, an associated ADC command (or stream of commands) is generated. The ADC Commands List Control Register (CLCR*x*) sets the assignment to an ADC command or to a stream of commands. When a trigger is linked to a timer or to the external trigger, a pulse with an active rising edge is generated. Additional features for the external triggers are available:

The external trigger output has:

- Pulse mode
- Toggle mode

In Toggle Mode, each trigger event is linked to the external trigger, the external trigger pin toggles. The ON-Time for both modes (Pulse mode and Toggle mode) of the triggers is defined from a COTR register (Control On Time Register). A guard time is also defined from the same register at the same value of the ON-Time. A new trigger will be generated only if the ON time + Guard Time has past. The ON-Time and the Guard Time are only used for external Triggers.

External signals can be asynchronous with motor control clock. For this reason a programmable digital filter is available. The external signal is considered at 1 if it is latched N time at 1, and is considered at 0 if it is latched N time at 0, where N is a value in the digital filter control register.

Trigger events in the SU can be initiated by hardware or by software, and an additional software control is possible for each trigger event (as for the MRS), so 1 bit for each trigger event in the CTU Control Register is used to generate an equivalent software trigger event. Each of these bits is cleared by a respective hardware or software trigger event.

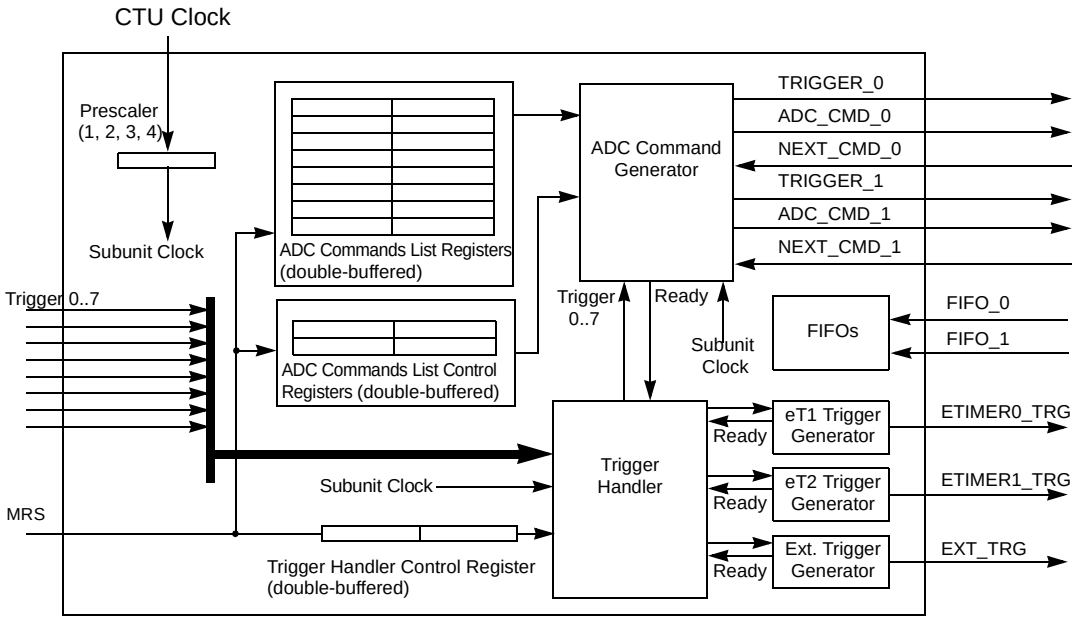


Figure 497. Scheduler subunit

26.4.1 ADC commands list

The ADC can be controlled by the CPU (CPU Control Mode) and by the CTU (CTU Control Mode). The CTU can control the ADC by sending an ADC command only when the ADC is in CTU control mode. During the CTU control mode, the CPU is able to write to the ADC registers but it can not start a new conversion. A control bit is allowed to select from the classic interface of the CTU control mode. Once selected, no change is possible unless a reset occurs.

The SU uses a Commands List in order to select the command to send to the ADC when a trigger event occurs. The commands list can hold twenty-four 16-bit commands (see *Section 26.4.2: ADC commands list format*) and it is double-buffered, that is, the commands list can be updated at any time between two consecutive MRS, but the changes become workable only after the next MRS occurs, and a correct reload is performed. In order to manage the commands list, 5 bits are available in the CLCRx (ADC Commands List Control Register *x*), for the position of the first command in the list of commands for each trigger event. The number of commands piloted by the same trigger event is defined directly in the commands list. For each command, a bit defines whether or not it is the first command of a commands list.

26.4.2 ADC commands list format

The CTU can be interfaced with two ADCs, supporting the Single Conversion Mode and the Dual Conversion Mode.

In Single Conversion Mode only one ADC starts a conversion at a time. In Dual Conversion Mode both ADCs start a conversion at the same time; in particular both the ADC conversions are performed at the same time while the storage of the results is performed in series. In Dual Conversion Mode, 4 bits select each channel number, and the conversion mode selection bit selects the Dual Conversion Mode. If the Single Conversion Mode is selected, 5 of the 8 bits reserved to select the channels in Dual Conversion Mode are re-used to select the channel (4 bits) and the ADC unit (1 bit). See *Section 26.8.10: Commands list register x (x = 1,...,24) (CLRx)*.

The result of each conversion is stored in one of the four available FIFOs.

The interrupt request bit is used as an interrupt request when ADC will complete the command with this bit set and it is only for CTU internal use. Before the next command to the CTU controls is sent, the value of the first command (FC) bit is checked to see if it is the current command is the first command of a new stream of consecutive commands or not. If not, the CTU sends the command.

According to the previous considerations, the commands in the list allow control on:

- Channel 0: number of ADC channel to sample from ADC unit 0 (4 bits)
- Channel 1: number of ADC channel to sample from ADC unit 1 (4 bits)
- FIFO selection bits for the ADC unit 0/1 (2 bits)
- Conversion Mode selection bit
- First command bit (only for CTU internal use)
- Interrupt request bit (only for CTU internal use)

On this device, only ADC_0 is implemented so a new CTU/ADC interface is implemented in order to interface the CTU and the only ADC_0. This new CTU/ADC interface is a logic machine between the CTU outputs and the ADC_0 inputs and it has no configuration registers. It is implemented to ensure software compatibility between SPC56xP60/54 and

the 512 Kbyte memory family device, in fact it is able to virtualize ADC_1 on SPC56xP60/54, so, for example, the user can write on SPC56xP60/54 a command to start a conversion on ADC_1 channels 0 and the CTU/ADC interface will translate this command into a command for ADC_0 channel 6. Moreover CTU/ADC interface will manage software programming mistakes for SPC56xP60/54 as not allowed Dual Conversion Mode or wrong ADC_0/1 channel number selection to ensure that the CTU does not go in a blocking status.

Table 474. ADC commands translation.

Input command	Output command
Single sampling ADC_0 channel 0	Single sampling ADC_0 channel 0
Single sampling ADC_0 channel 1	Single sampling ADC_0 channel 1
Single sampling ADC_0 channel 2	Single sampling ADC_0 channel 2
Single sampling ADC_0 channel 3	Single sampling ADC_0 channel 3
Single sampling ADC_0 channel 4	Single sampling ADC_0 channel 4
Single sampling ADC_0 channel 5	Single sampling ADC_0 channel 5
Single sampling ADC_0 channel 6	Single sampling ADC_0 channel 6
Single sampling ADC_0 channel 7	Single sampling ADC_0 channel 7
Single sampling ADC_0 channel 8	Single sampling ADC_0 channel 8
Single sampling ADC_0 channel 9	Single sampling ADC_0 channel 9
Single sampling ADC_0 channel 10	Single sampling ADC_0 channel 10
Single sampling ADC_0 channel 11	Single sampling ADC_0 channel 11
Single sampling ADC_0 channel 12	Single sampling ADC_0 channel 12
Single sampling ADC_0 channel 13	Single sampling ADC_0 channel 13
Single sampling ADC_0 channel 14	Single sampling ADC_0 channel 14
Single sampling ADC_0 channel 15	Single sampling ADC_0 channel 15
Single sampling ADC_1 Channel 0	Single sampling ADC_0 Channel 16
Single sampling ADC_1 Channel 1	Single sampling ADC_0 Channel 17
Single sampling ADC_1 Channel 2	Single sampling ADC_0 Channel 18
Single sampling ADC_1 Channel 3	Single sampling ADC_0 Channel 19
Single sampling ADC_1 Channel 4	Single sampling ADC_0 Channel 20
Single sampling ADC_1 Channel 5	Single sampling ADC_0 Channel 21
Single sampling ADC_1 Channel 6	Single sampling ADC_0 Channel 22
Single sampling ADC_1 Channel 7	Single sampling ADC_0 Channel 23
Single sampling ADC_1 Channel 8	Single sampling ADC_0 Channel 24
Single sampling ADC_1 Channel 9	Single sampling ADC_0 Channel 25
Single sampling ADC_1 Channel 10	Single sampling ADC_0 Channel 26
Single sampling ADC_1 Channel 11	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 12	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 13	Not valid - force EOC to CTU

Table 474. ADC commands translation.

Input command	Output command
Single sampling ADC_1 Channel 14	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 15	Not valid - force EOC to CTU
Dual sampling ADC_0 channel x / ADC_1 channel y	Not valid - force EOC to CTU

26.4.3 ADC results

ADC results can be stored in the channel relevant standard result register and/or in one of the four FIFOs: the different FIFOs allow to dispatch ADC results according to their type of acquisition (for example phase currents, rotor position or ground-noise). Each FIFO has its own interrupt line and DMA request signal (plus an individual overflow error bit in the FIFO status register). The store location is specified in the ADC command, that is, the FIFOs are available only in CTU Control Mode. Each entry of a FIFO is 32-bits.

The size of the FIFOs are the following:

- FIFO1 and FIFO2—16 entries
- FIFO3 and FIFO4—4 entries (low acquisition rate FIFOs)

Results in each FIFO can be read by a 16-bit read transaction (only the result is read in order to minimize the CPU load before computing on results) or by a 32-bit read transaction (both the result and the channel number are read in order to avoid blind acquisitions), 5 bits in the upper 16 bits indicate the ADC unit (1 bit) and the channel number (4 bits). The result registers (only for the FIFOs) can be read from two different addresses in the ADC memory map. The format of the result depends on the address from which it is read. The available formats are:

- Unsigned right-justified

  Conversion result is unsigned right-justified data. Bits [9:0] are used for 10-bit resolution and bits [15:10] always return zero when read.
- Signed left-justified

  Conversion result is signed left-justified data. Bit [15] is reserved for sign and is always read as zero for this ADC, bits [14:5] are used for 10-bit resolution, and bits [4:0] always return zero when read.

26.5 Reload mechanism

Some CTU registers are double-buffered, and the reload is controlled by a reload enable bit, as the TGSISR_RE bit or the DFE bit, but for the most of the double-buffered registers, the reload is controlled by the MRS occurrence, and it is synchronized with the beginning of the CTU control period.

If the MRS occurs while the user is updating some double-buffered registers, eg. some registers of the triggers list, the new triggers list will be a mix of the old triggers list and the new triggers list, because the user has not ended the update of the triggers list before the MRS occurrence.

In order to avoid this case, 1 bit enables the reload operation, that is, to inform the CTU that the user has ended updates to the double-buffered registers, and the reload can be performed without problems of mixed scenarios. In order to guarantee the coherency, the

reload of all double-buffered registers is enabled by setting GRE (General Reload Enable) bit in the CTU Control Register. The user must ensure that all intended double-buffered registers are updated before a new MRS occurrence. If an MRS occurs before a GRE bit is set (for example, wrong application timing), the update is not performed, the previous values of all double-buffered registers remain active, the error flag is set (the MRS_RE bit in the CTU Error Flag Register) and, if enabled, CTU performs an interrupt request.

All the double-buffered registers use the General Reload Enable (GRE) bit to enable the reload when the MRS occurs. The GRE bit is R/S (Read/Set) and if this bit is 1, the reload can be performed, while if this bit is 0, the reload is not performed. A correct reload resets the GRE bit. None of the double-buffered registers can be written while the GRE bit remains set. The GRE bit can be reset by the occurrence of the next MRS (that is, a correct reload) or by software setting the CGRE bit.

The CGRE is reset by hardware after that GRE bit is reset. If the user sets the CGRE bit and at the same time a MRS occurs, CGRE has the priority so GRE is reset and the reload is not performed. In the same way, the GRE has the priority when compared with the MRS occurrence, and the CGRE has the priority compared with the GRE (the two bits are in the same register so they can be set in the same time). MRS has the priority compared with the re-synchronization bit of the TGSISR.

In order to verify if a reload error occurs, FGRE (Flag GRE bit in the CTU Control Register) bit is used. When one of the double-buffered registers is written, this flag is set to 1 and it is reset by a correct reload. When the MRS occurs while FGRE is 1 and GRE is 1, a correct reload is performed (because all intended registers have been updated before the MRS occurs). If FGRE is 1 and GRE is 0, a reload is not performed, the error flag (MRS_RE) is set and (if enabled) an interrupt for an error is performed (in this case at least one register was written but the update has not ended before the MRS occurrence). If FGRE is 0 it is not necessary to perform a reload because all the double-buffered registers are unchanged (see *Figure 498*).

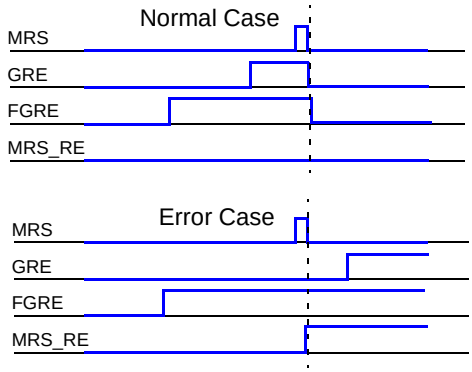


Figure 498. Reload error scenario

26.6 Power safety mode

To reduce power consumption two mechanisms are implemented:

- MDIS bit in the CTUPCR
- STOP mode

26.6.1 MDIS bit

The MDIS bit in the CTUPCR is used for stopping the clock to all non memory mapped registers.

26.6.2 STOP mode

To reduce consumption, it is also possible to enable a stop request from the Mode Entry module. The FIFOs are considered a lot like memory mapped registers, otherwise there could be some problems if a read operation occurs during the MDIS bit set period. When the clock is started after an MDIS bit setting or a stop signal, some mistakes could occur. For example, a wrong trigger could be provided because it was programmed before the stop signal was performed, and some incorrect write operations into the FIFOs could happen. For this reason after a stop signal after a MDIS bit setting the FIFO have to be empty. In order to avoid the problems linked to a wrong trigger, the CTU output can be disabled by the CTU_ODIS bit and the ADC interface state machine can be reset by the CRU_ADC_R (see *Section 26.8.21: Cross triggering unit control register (CTUCR)*).

26.7 Interrupts and DMA requests

26.7.1 DMA support

The DMA can be used to configure the CTU registers. One DMA channel is reserved for performing a block transfer, and the MRS can be used as an optional DMA request signal (MRS_DMAE bit in the CTU Interrupt/DMA Register).

Note: If enabled, the DMA request on the MRS occurrence is performed only if a reload is performed, that is, only if the GRE bit is set.

Moreover, this CTU implementation requires DMA support for reading the data from the FIFOs. One DMA channel is available for each FIFO. Each FIFO can perform a DMA request when the number of words stored in the FIFO reaches the threshold value.

26.7.2 CTU faults and errors

Faults and errors that could occur during the programming include:

- An MRS occurs while user is updating the double-buffered registers and the MRS_RE bit is set.
- Receiving more than eight EVs before that the next MRS occurs in TGS sequential mode and the SM_TO bit is set.
- A trigger event occurs during the time when the actions of the previous trigger event are not completed (user ensures no trigger event occurs during another one is processed, but if user makes a mistake and a trigger event occurs when another one is processed, the incoming trigger event will be lost and an error occurs).

  There are four overrun flags (one for each type of output). The general mechanism shall be as in *Figure 497*.

  The Trigger Handler, when a trigger event occurs, and the corresponding Ready signal is high, presents the respective trigger signal (one cycle high time + one cycle low time) to the respective generator sub-block (ADC Command Generator, eT0 Trigger Generator, eT1 Trigger Generator or Ext. Trigger Generator). This generator sub-block

then generates the requested signal. Until this real signal is generated (including guard time) the Ready signal is kept low.

In the case of ADC command generator, the Ready signal shall be kept low until the last conversion in the batch is finished. The respective overrun flag is set at the following conditions:

– Ready signal is low.

– The rising edge of the respective trigger signal (from Trigger Handler to generator sub-block) occurs.

This architecture allows the user to pre-set a trigger to the eTimer0 in the middle of an ADC conversion, that is, the SU will be considered busy only if a request to perform the same action that the SU is already performing occurs. One of the following bits is set: ADC_OE, T0_OE, T1_OE, or ET_OE.

- Invalid (unrecognized) ADC command and the ICE bit is set.
- The MRS occurs before the enabled trigger events occur and the MRS_O bit is set.
- TGS overrun in sequential mode: a new incoming EV occurs before than the trigger event selected by the previous EV occurs. The incoming EV sets an internal busy flag. The outgoing trigger event (all line are ORed) resets this flag to 0. TGS Overrun in the sequential mode shall be generated under the following conditions:
  – TGS is in sequential mode
  – there is an incoming EV while the busy flag is high. the TGS_OSM bit is set.

The faults/errors flags in the CTU error flag register and in the CTU interrupt flag register can be cleared by writing a 1 while writing a 0 has no effect. The CTU does not support a write-protection mechanism.

26.7.3 CTU interrupt/DMA requests

The CTU can perform the following interrupt/DMA requests (15 interrupt lines):

- Error interrupt request (see *Section 26.7.2: CTU faults and errors*) (1 interrupt line)
- ADC command interrupt request (1 interrupt line)
- Interrupt request on MRS occurrence (1 interrupt line)
- Interrupt request on each trigger event occurrence (1 interrupt line for each trigger event)
- FIFOs interrupt requests and/or DMA transfer request (1 interrupt line for each FIFO)
- DMA transfer request on the MRS occurrence if GRE bit is set

The interrupt flags are shown in *Table 475*.

Table 475. CTU interrupts

Category	Interrupt	Interrupt function
Managed individually	MRS_I	MRS Interrupt flag (IRQ193)
	T0_I	Trigger 0 interrupt flag (IRQ194)
	T1_I	Trigger 1 interrupt flag (IRQ195)
	T2_I	Trigger 2 interrupt flag (IRQ196)
	T3_I	Trigger 3 interrupt flag (IRQ197)
	T4_I	Trigger 4 interrupt flag (IRQ198)
	T5_I	Trigger 5 interrupt flag (IRQ199)
	T6_I	Trigger 6 interrupt flag (IRQ200)
	T7_I	Trigger 7 interrupt flag (IRQ201)
	ADC_I	ADC command interrupt flag (IRQ206)
ORed onto FIFO1_I (IRQ202)	FIFO_FULL0	This bit is set to 1 if the FIFO 0 is full.
	FIFO_EMPTY0	This bit is set to 1 if the FIFO 0 is empty.
	FIFO_OVERFLOW0	This bit is set to 1 if the number of words exceeds the value set in the threshold 0.
	FIFO_OVERRUN0	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL0 flag is set.
ORed onto FIFO2_I (IRQ203)	FIFO_FULL1	This bit is set to 1 if the FIFO 1 is full.
	FIFO_EMPTY1	This bit is set to 1 if the FIFO 1 is empty.
	FIFO_OVERFLOW1	This bit is set to 1 if the number of words exceeds the value set in the threshold 1.
	FIFO_OVERRUN1	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL1 flag is set.
ORed onto FIFO3_I (IRQ204)	FIFO_FULL2	This bit is set to 1 if the FIFO 2 is full.
	FIFO_EMPTY2	This bit is set to 1 if the FIFO 2 is empty.
	FIFO_OVERFLOW2	This bit is set to 1 if the number of words exceeds the value set in the threshold 2.
	FIFO_OVERRUN2	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL2 flag is set.
ORed onto FIFO4_I (IRQ205)	FIFO_FULL3	This bit is set to 1 if the FIFO 3 is full.
	FIFO_EMPTY3	This bit is set to 1 if the FIFO 3 is empty.
	FIFO_OVERFLOW3	This bit is set to 1 if the number of words exceeds the value set in the threshold 3.
	FIFO_OVERRUN3	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL3 flag is set.

Table 475. CTU interrupts (continued)

Category	Interrupt	Interrupt function
ORed onto ERR_I (IRQ207)	MRS_RE	Master Reload Signal Reload Error
	SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode
	ICE	Invalid Command Error
	MRS_O	Master Reload Signal Overrun
	TGS_OSM	TGS Overrun in Sequential Mode
	ADC_OE	ADC command generation Overrun Error
	T0_OE	Timer 0 trigger generation Overrun Error
	T1_OE	Timer 1 trigger generation Overrun Error
	ET_OE	External Trigger generation Overrun Error

26.8 Memory map

Table 476. CTU memory map

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0000	TGSISR — Trigger Generator Subunit Input Selection Register	*on page 859*
0x0004	TGSCRR — Trigger Generator Subunit Control Register	*on page 861*
0x0006	T0CR — Trigger 0 Compare Register	*on page 862*
0x0008	T1CR — Trigger 1 Compare Register	*on page 862*
0x000A	T2CR — Trigger 2 Compare Register	*on page 862*
0x000C	T3CR — Trigger 3 Compare Register	*on page 862*
0x000E	T4CR — Trigger 4 Compare Register	*on page 862*
0x0010	T5CR — Trigger 5 Compare Register	*on page 862*
0x0012	T6CR — Trigger 6 Compare Register	*on page 862*
0x0014	T7CR — Trigger 7 Compare Register	*on page 862*
0x0016	TGSCCR — TGS Counter Compare Register	*on page 862*
0x0018	TGSCRR — TGS Counter Reload Register	*on page 863*
0x001A	Reserved	
0x001C	CLCR1 — Commands List Control Register 1	*on page 863*
0x0020	CLCR2 — Commands List Control Register 2	*on page 864*
0x0024	THCR1 — Trigger Handler Control Register 1	*on page 864*
0x0028	THCR2 — Trigger Handler Control Register 2	*on page 866*
0x002C	CLR1—Commands List Register 1	*on page 868*
0x002E	CLR2—Commands List Register 2	*on page 868*

Table 476. CTU memory map (continued)

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0030	CLR3—Commands List Register 3	on page 868
0x0032	CLR4—Commands List Register 4	on page 868
0x0034	CLR5—Commands List Register 5	on page 868
0x0036	CLR6—Commands List Register 6	on page 868
0x0038	CLR7—Commands List Register 7	on page 868
0x003A	CLR8—Commands List Register 8	on page 868
0x003C	CLR9—Commands List Register 9	on page 868
0x003E	CLR10—Commands List Register 10	on page 868
0x0040	CLR11—Commands List Register 11	on page 868
0x0042	CLR12—Commands List Register 12	on page 868
0x0044	CLR13—Commands List Register 13	on page 868
0x0046	CLR14—Commands List Register 14	on page 868
0x0048	CLR15—Commands List Register 15	on page 868
0x004A	CLR16—Commands List Register 16	on page 868
0x004C	CLR17—Commands List Register 17	on page 868
0x004E	CLR18—Commands List Register 18	on page 868
0x0050	CLR19—Commands List Register 19	on page 868
0x0052	CLR20—Commands List Register 20	on page 868
0x0054	CLR21—Commands List Register 21	on page 868
0x0056	CLR22—Commands List Register 22	on page 868
0x0058	CLR23—Commands List Register 23	on page 868
0x005A	CLR24—Commands List Register 24	on page 868
0x005C—0x006B	Reserved	
0x006C	FDCR — FIFO DMA Control Register	on page 869
0x0070	FCR — FIFO Control Register	on page 870
0x0074	FTH — FIFO Threshold Register	on page 872
0x0078–0x007B	Reserved	
0x007C	FST — FIFO Status Register	on page 872
0x0080	FR0 — FIFO Right aligned data register 0	on page 874
0x0084	FR1 — FIFO Right aligned data register 1	on page 874
0x0088	FR2 — FIFO Right aligned data register 2	on page 874
0x008C	FR3 — FIFO Right aligned data register 3	on page 874
0x0090–0x009F	Reserved	

Table 476. CTU memory map (continued)

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x00A0	FL0 — FIFO Left aligned data register 0	on page 875
0x00A4	FL1 — FIFO Left aligned data register 1	on page 875
0x00A8	FL2 — FIFO Left aligned data register 2	on page 875
0x00AC	FL3 — FIFO Left aligned data register 3	on page 875
0x00B0–0x00BF	Reserved	
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	on page 875
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	on page 876
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	on page 877
0x00C6	COTR — Control ON-Time Register	on page 878
0x00C8	CTUCR — Cross triggering unit control register	on page 879
0x00CA	CTUDF — Cross Triggering Unit Digital Filter register	on page 880
0x00CC	CTUPCR — Cross Triggering Unit Power Control Register	on page 880
0x00CE–0x3FFF	Reserved	

Table 477. TGS registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x0000	TGSISR — Trigger Generator Subunit Input Selection Register	Yes	TGSISR_RE	0x0000_0000
0x0004	TGSCRR — Trigger Generator Subunit Control Register	Yes	MRS	0x0000
0x0006	T0CR — Trigger 0 Compare Register	Yes	MRS	0x0000
0x0008	T1CR — Trigger 1 Compare Register	Yes	MRS	0x0000
0x000A	T2CR — Trigger 2 Compare Register	Yes	MRS	0x0000
0x000C	T3CR — Trigger 3 Compare Register	Yes	MRS	0x0000
0x000E	T4CR — Trigger 4 Compare Register	Yes	MRS	0x0000
0x0010	T5CR — Trigger 5 Compare Register	Yes	MRS	0x0000
0x0012	T6CR — Trigger 6 Compare Register	Yes	MRS	0x0000
0x0014	T7CR — Trigger 7 Compare Register	Yes	MRS	0x0000
0x0016	TGSCCR — TGS Counter Compare Register	Yes	MRS	0x0000
0x0018	TGSCRR — TGS Counter Reload Register	Yes	MRS	0x0000

Table 478. SU registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x001C	CLCR1 — Commands List Control Register 1	Yes	MRS	0x0000_0000
0x0020	CLCR2 — Commands List Control Register 2	Yes	MRS	0x0000_0000
0x0024	THCR1 — Trigger Handler Control Register 1	Yes	MRS	0x0000_0000
0x0028	THCR2 — Trigger Handler Control Register 2	Yes	MRS	0x0000_0000
0x002C – 0x005A	CLRx — Commands List Register x (x = 1,...,24)	Yes	MRS	0x0000

Table 479. CTU registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	No	—	0x0000
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	No	—	0x0000
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	No	—	0x0000
0x00C6	COTR — Control ON-Time Register	Yes	MRS	0x0000
0x00C8	CTUCR — Cross triggering unit control register	No	—	0x0000
0x00CA	CTUDF — Cross Triggering Unit Digital Filter	Yes	DFE	0x0000
0x00CC	CTUPCR — Cross Triggering Unit Power Control Register	No	—	0x0000

Table 480. FIFO registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x006C	FDCR — FIFO DMA Control Register	No	—	0x0000
0x0070	FCR — FIFO Control Register	No	—	0x0000_0000
0x0074	FTH — FIFO Threshold Register	No	—	0x0000_0000
0x007C	FST — FIFO Status Register	No	—	0x0000_0000
0x0080	FR0 — FIFO Right aligned data 0	No	—	0x0000_0000
0x0084	FR1 — FIFO Right aligned data 1	No	—	0x0000_0000
0x0088	FR2 — FIFO Right aligned data 2	No	—	0x0000_0000
0x008C	FR3 — FIFO Right aligned data 3	No	—	0x0000_0000
0x00A0	FL0 — FIFO Left aligned data 0	No	—	0x0000_0000
0x00A4	FL1 — FIFO Left aligned data 1	No	—	0x0000_0000
0x00A8	FL2 — FIFO Left aligned data 2	No	—	0x0000_0000
0x00AC	FL3 — FIFO Left aligned data 3	No	—	0x0000_0000

26.8.1 Trigger Generator Sub-unit Input Selection Register (TGSISR)

Figure 499. Trigger Generator Sub-unit Input Selection Register (TGSISR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	I15_FE	I15_RE	I14_FE	I14_RE	I13_FE	I13_RE	I12_FE	I12_RE	I11_FE	I11_RE	I10_FE	I10_RE	I9_FE	I9_RE	I8_FE	I8_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	I7_FE	I7_RE	I6_FE	I6_RE	I5_FE	I5_RE	I4_FE	I4_RE	I3_FE	I3_RE	I2_FE	I2_RE	I1_FE	I1_RE	I0_FE	I0_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 481. TGSISR field descriptions

Field	Description
I15_FE	Input 15 — ext_signals Falling edge Enable 0 Disabled 1 Enabled
I15_RE	Input 15 — ext_signals Rising edge Enable 0 Disabled 1 Enabled
I14_FE	Input 14 — eTimer1 [ETC2] Falling edge Enable 0 Disabled 1 Enabled
I14_RE	Input 14 — eTimer1 [ETC2] Rising edge Enable 0 Disabled 1 Enabled
I13_FE	Input 13 — eTimer0 [ETC2] Falling edge Enable 0 Disabled 1 Enabled
I13_RE	Input 13 — eTimer0 [ETC2] Rising edge Enable 0 Disabled 1 Enabled
I12_FE	Input 12 —PWM X[3] Falling edge Enable 0 Disabled 1 Enabled
I12_RE	Input 12 — PWM X[3] Rising edge Enable 0Disabled 1 Enabled
I11_FE	Input 11 — PWM X[2] Falling edge Enable 0 Disabled 1 Enabled

Table 481. TGSISR field descriptions (continued)

Field	Description
I11_RE	Input 11 — PWM X[2] Rising edge Enable 0 Disabled 1 Enabled
I10_FE	Input 10 — PWM X[1] Falling edge Enable 0Disabled 1 Enabled
I10_RE	Input 10 — PWM X[1] Rising edge Enable 0 Disabled 1 Enabled
I9_FE	Input 9 — PWM X[0] Falling edge Enable 0Disabled 1 Enabled
I9_RE	Input 9 — PWM X[0] Rising edge Enable 0 Disabled 1 Enabled
I8_FE	Input 8 — PWM OUT_TRIG 1 [3] Falling edge Enable 0 Disabled 1 Enabled
I8_RE	Input 8 — PWM OUT_TRIG 1 [3] Rising edge Enable 0 Disabled 1 Enabled
I7_FE	Input 7 — PWM OUT_TRIG 1 [2] Falling edge Enable 0 Disabled 1 Enabled
I7_RE	Input 7 — PWM OUT_TRIG 1 [2] Rising edge Enable 0 Disabled 1 Enabled
I6_FE	Input 6 — PWM OUT_TRIG 1 [1] Falling edge Enable 0 Disabled 1 Enabled
I6_RE	Input 6 — PWM OUT_TRIG 1 [1] Rising edge Enable 0 Disabled 1 Enabled
I5_FE	Input 5 — PWM OUT_TRIG 1 [0] Falling edge Enable 0 Disabled 1 Enabled
I5_RE	Input 5 — PWM OUT_TRIG 1 [0] Rising edge Enable 0 Disabled 1 Enabled
I4_FE	Input 4 — PWM OUT_TRIG 0 [3] Falling edge Enable 0 Disabled 1 Enabled

Table 481. TGSISR field descriptions (continued)

Field	Description
I4_RE	Input 4 — PWM OUT_TRIG 0 [3] Rising edge Enable 0 Disabled 1 Enabled
I3_FE	Input 3 — PWM OUT_TRIG 0 [2] Falling edge Enable 0 Disabled 1 Enabled
I3_RE	Input 3 — PWM OUT_TRIG 0 [2] Rising edge Enable 0 Disabled 1 Enabled
I2_FE	Input 2 — PWM OUT_TRIG 0 [1] Falling edge Enable 0 Disabled 1 Enabled
I2_RE	Input 2 — PWM OUT_TRIG 0 [1] Rising edge Enable 0 Disabled 1 Enabled h
I1_FE	Input 1 — PWM OUT_TRIG 0 [0] Falling edge Enable 0 Disabled 1 Enabled
I1_RE	Input 1 — PWM OUT_TRIG 0 [0] Rising edge Enable 0 Disabled 1 Enabled
I0_FE	Input 0 — PWM Reload Falling edge Enable 0 Disabled 1 Enabled
I0_RE	Input 0 — PWM Reload Rising edge Enable 0 Disabled 1 Enabled

26.8.2 Trigger Generator Sub-unit Control Register (TGSCR)

Figure 500. Trigger Generator Sub-unit Control Register (TGSCR)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	ET_TM	PRES			MRS_SM				TGS_M
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 482. TGSCR field descriptions

Field	Description
ET_TM	This bit enables toggle mode for external triggers.
PRES	TGS and SU prescaler selection bits 00 1 01 2 10 3 11 4
MRS_SM	Master Reload Selection in Sequential Mode (5 bits to select one of 32 inputs)
TGS_M	Trigger Generator Subunit Mode 0 Triggered Mode 1 Sequential Mode

26.8.3 Trigger x Compare Register (TxCR, *x* = 0...7)

Figure 501. Trigger x Compare Register (TxCR, *x* = 0...7)

Address: Base + 0x0006 (T0CR), Base + 0x0008 (T1CR), Base + 0x000A (T2CR), Base + 0x000C (T3CR), Base + 0x000E (T4CR), Base + 0x0010 (T5CR), Base + 0x0012 (T6CR), Base + 0x0014 (T7CR)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TxCRV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 483. TxCR field descriptions

Field	Description
TxCRV	Trigger x Compare Register Value

26.8.4 TGS Counter Compare Register (TGSCCR)

Figure 502. TGS Counter Compare Register (TGSCCR)

Address: Base + 0x0016

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TGSCCV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 484. TGSCCR field format

Field	Description
TGSCCV	TGS Counter Compare Value

26.8.5 TGS Counter Reload Register (TGSCRR)

Figure 503. TGS Counter Reload Register (TGSCRR)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TGSCRV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 485. TGSCRR field descriptions

Field	Description
TGSCRV	TGS Counter Reload Value

26.8.6 Commands list control register 1 (CLCR1)

Figure 504. Commands list control register 1 (CLCR1)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T3_INDEX					0	0	0	T2_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T1_INDEX					0	0	0	T0_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 486. CLCR1 field descriptions

Field	Description
T3_INDEX	Trigger 3 Commands List first command address
T2_INDEX	Trigger 2 Commands List first command address
T1_INDEX	Trigger 1 Commands List first command address
T0_INDEX	Trigger 0 Commands List first command address

26.8.7 Commands list control register 2 (CLCR2)

Figure 505. Commands list control register 2 (CLCR2)

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T7_INDEX					0	0	0	T6_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T5_INDEX					0	0	0	T4_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 487. CLCR2 field descriptions

Field	Description
T7_INDEX	Trigger 7 Commands List first command address
T6_INDEX	Trigger 6 Commands List first command address
T5_INDEX	Trigger 5 Commands List first command address
T4_INDEX	Trigger 4 Commands List first command address

26.8.8 Trigger handler control register 1 (THCR1)

Figure 506. Trigger handler control register 1 (THCR1)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T3_E	T3_ ETE	T3_ T1E	T3_ T0E	T3_ ADCE	0	0	0	T2_E	T2_ ETE	T2_ T1E	T2_ T0E	T2_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T1_E	T1_ ETE	T1_ T1E	T1_ T0E	T1_ ADCE	0	0	0	T0_E	T0_ ETE	T0_ T1E	T0_ T0E	T0_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 488. THCR1 field descriptions

Field	Description
T3_E	Trigger 3 enable 0 Disabled 1 Enabled
T3_ETE	Trigger 3 External Trigger output enable 0 Disabled 1 Enabled
T3_T1E	Trigger 3 Timer 1 output enable 0 Disabled 1 Enabled
T3_T0E	Trigger 3 Timer 0 output enable 0 Disabled 1 Enabled
T3_ADCE	Trigger 3 ADC command output enable 0 Disabled 1 Enabled
T2_E	Trigger 2 enable 0 Disabled 1 Enabled
T2_ETE	Trigger 2 External Trigger output enable 0 Disabled 1 Enabled
T2_T1E	Trigger 2 Timer 1 output enable 0 Disabled 1 Enabled
T2_T0E	Trigger 2 Timer 0 output enable 0 Disabled 1 Enabled
T2_ADCE	Trigger 2 ADC command output enable 0 Disabled 1 Enabled
T1_E	Trigger 1 enable 0 Disabled 1 Enabled
T1_ETE	Trigger 1 External Trigger output enable 0 Disabled 1 Enabled
T1_T1E	Trigger 1 Timer 1 output enable 0 Disabled 1 Enabled
T1_T0E	Trigger 1 Timer 0 output enable 0 Disabled 1 Enabled

Table 488. THCR1 field descriptions (continued)

Field	Description
T1_ADCE	Trigger 1 ADC command output enable 0 Disabled 1 Enabled
T0_E	Trigger 0 enable 0 Disabled 1 Enabled
T0_ETE	Trigger 0 External Trigger output enable 0 Disabled 1 Enabled
T0_T1E	Trigger 0 Timer 1 output enable 0 Disabled 1 Enabled
T0_T0E	Trigger 0 Timer 0 output enable 0 Disabled 1 Enabled
T0_ADCE	Trigger 0 ADC command output enable 0 Disabled 1 Enabled

26.8.9 Trigger handler control register 2 (THCR2)

Figure 507. Trigger handler control register 2 (THCR2)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T7_E	T7_ ETE	T7_ T1E	T7_ T0E	T7_ ADCE	0	0	0	T6_E	T6_ ETE	T6_ T1E	T6_ T0E	T6_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T5_E	T5_ ETE	T5_ T5E	T5_ T0E	T5_ ADCE	0	0	0	T4_E	T4_ ETE	T4_ T1E	T4_ T0E	T4_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 489. THCR2 field descriptions

Field	Description
T7_E	Trigger 7 enable 0 Disabled 1 Enabled
T7_ETE	Trigger 7 External Trigger output enable 0 Disabled 1 Enabled

Table 489. THCR2 field descriptions (continued)

Field	Description
T7_T1E	Trigger 7 Timer 1 output enable 0 Disabled 1 Enabled
T7_T0E	Trigger 7 Timer 0 output enable 0 Disabled 1 Enabled
T7_ADCE	Trigger 7 ADC command output enable 0 Disabled 1 Enabled
T6_E	Trigger 6 enable 0 Disabled 1 Enabled
T6_ETE	Trigger 6 External Trigger output enable 0 Disabled 1 Enabled
T6_T1E	Trigger 6 Timer 1 output enable 0 Disabled 1 Enabled
T6_T0E	Trigger 6 Timer 0 output enable 0 Disabled 1 Enabled
T6_ADCE	Trigger 6 ADC command output enable 0 Disabled 1 Enabled
T5_E	Trigger 5 enable 0 Disabled 1 Enabled
T5_ETE	Trigger 5 External Trigger output enable 0 Disabled 1 Enabled
T5_T1E	Trigger 5 Timer 1 output enable 0 Disabled 1 Enabled
T5_T0E	Trigger 5 Timer 0 output enable 0 Disabled 1 Enabled
T5_ADCE	Trigger 5 ADC command output enable 0 Disabled 1 Enabled
T4_E	Trigger 4 enable 0 Disabled 1 Enabled

Table 489. THCR2 field descriptions (continued)

Field	Description
T4_ETE	Trigger 4 External Trigger output enable 0 Disabled 1 Enabled
T4_T1E	Trigger 4 Timer 1 output enable 0 Disabled 1 Enabled
T4_T0E	Trigger 4 Timer 0 output enable 0 Disabled 1 Enabled
T4_ADCE	Trigger 4 ADC command output enable 0 Disabled 1 Enabled

26.8.10 Commands list register x (x = 1,...,24) (CLR*x*)

Figure 508 and *Table 490* show the register configured for ADC command format in single conversion mode (CMS = 0) (CLR*x*).

Figure 508. Commands list register *x* (*x* = 1,...,24) (CMS = 0)

Address: Base + 0x002C,...,0x005A (See *Table 476*)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	FC	CMS	0	FIFO		0	0	0	0	SU	0	CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 490. CLR*x* (CMS = 0) field descriptions

Field	Description
CIR	Command Interrupt Request bit 0 Disabled 1 Enabled
FC	First command bit 0 Not first command 1 First command
CMS	Conversion mode selection bit 0 Single conversion mode 1 Dual conversion mode
FIFO	FIFO for ADC unit 0/1 00 FIFO 0 selected 01 FIFO 1 selected 10 FIFO 2 selected 11 FIFO 3 selected

Table 490. CLR*x* (CMS = 0) field descriptions (continued)

Field	Description
SU	Selection Unit bit 0 ADC unit 0 selected 1 ADC unit 1 selected
CH	ADC unit channel number

Figure 509 and *Table 491* show the register configured for ADC command format in dual conversion mode (CMS = 1).

Figure 509. Commands list register x (x = 1,...,24) (CMS = 1)

Address: Base + 0x002C ... 0x005A (See *Table 476*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	FC	CMS	0	FIFO		0	CH_B				0	CH_A			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 491. CLRx (CMS = 1) field descriptions

Field	Description
CIR	Command Interrupt Request bit 0 Disabled 1 Enabled
FC	First command bit 0 Not first command 1 First command
CMS	Conversion mode selection 0 Single conversion mode 1 Dual conversion mode
FIFO	FIFO for ADC unit A/B
CH_B	ADC unit B channel number
CH_A	ADC unit A channel number

26.8.11 FIFO DMA control register (FDCR)

Figure 510. FIFO DMA control register (FDCR)

Address: Base + 0x006C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	DE3	DE2	DE1	DE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 492. FDCR field descriptions

Name	Description
DEx	This bit enables DMA for the FIFOx 0 Disabled 1 Enabled

26.8.12 FIFO control register (FCR)

Figure 511. FIFO control register (FCR)

Address: Base + 0x0070 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR_EN3	OF_EN3	EMPTY_EN3	FULL_EN3	OR_EN2	OF_EN2	EMPTY_EN2	FULL_EN2	OR_EN1	OF_EN1	EMPTY_EN1	FULL_EN1	OR_EN0	OF_EN0	EMPTY_EN0	FULL_EN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 493. FCR field descriptions

Field	Description
OR_EN3	FIFO 3 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN3	FIFO 3 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN3	FIFO 3 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN3	FIFO 3 Full interrupt enable 0 Disabled 1 Enabled
OR_EN2	FIFO 2 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN2	FIFO 2 threshold Overflow interrupt enable 0 Disabled 1 Enabled

Table 493. FCR field descriptions (continued)

Field	Description
EMPTY_EN2	FIFO 2 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN2	FIFO 2 Full interrupt enable 0 Disabled 1 Enabled
OR_EN1	FIFO 1 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN1	FIFO 1 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN1	FIFO 1 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN1	FIFO 1 Full interrupt enable 0 Disabled 1 Enabled
OR_EN0	FIFO 0 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN0	FIFO 0 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN0	FIFO 0 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN0	FIFO 0 Full interrupt enable 0 Disabled 1 Enabled

26.8.13 FIFO threshold register (FTH)

Figure 512. FIFO threshold register (FTH)

Address: Base + 0x0074 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TH3								TH2							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TH1								TH0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 494. FTH field descriptions

Field	Description
TH3	FIFO 3 Threshold
TH2	FIFO 2 Threshold
TH1	FIFO 1 Threshold
TH0	FIFO 0 Threshold

26.8.14 FIFO status register (FST)

Figure 513. FIFO status register (FST)

Address: Base + 0x007C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR3	OF3	EMP3	FULL3	OR2	OF2	EMP2	FULL2	OR1	OF1	EMP1	FULL1	OR0	OF0	EMP0	FULL0
W	r1c				r1c				r1c				r1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 495. FST field descriptions

Field	Description
OR3	FIFO 3 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF3	FIFO 3 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP3	FIFO 3 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL3	FIFO 3 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR2	FIFO 2 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF2	FIFO 2 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP2	FIFO 2 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL2	FIFO 2 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR1	FIFO 1 Overrun interrupt flag A read of this bit clears it. 0Interrupt has not occurred. 1 Interrupt has occurred.
OF1	FIFO 1 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP1	FIFO 1 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL1	FIFO 1 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR0	FIFO 0 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 495. FST field descriptions (continued)

Field	Description
OF0	FIFO 0 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP0	FIFO 0 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL0	FIFO 0 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

26.8.15 FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*)

Figure 514. FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*)

Address: Base + 0x0080,...,0x008C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	N_CH[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	DATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 496. FR*x* field descriptions

Field	Description
N_CH[4:0]	Number of stored channel 0xxxx: Result comes from an ADC_1 channel 1xxxx: Result comes from an ADC_0 channel
DATA	Data of stored channel

26.8.16 FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*)

Figure 515. FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*)

Address: Base + 0x00A0,...,0x00AC Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	N_CH[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	DATA										0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 497. FL*x* field descriptions

Field	Description
N_CH[4:0]	Number of stored channel 0xxxx: Result comes from an ADC_1 channel 1xxxx: Result comes from an ADC_0 channel
DATA	Data of stored channel

26.8.17 Cross triggering unit error flag register (CTUEFR)

Figure 516. Cross triggering unit error flag register (CTUEFR)

Address: Base + 0x00C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	ET_OE	T1_OE	T0_OE	ADC_OE	TGS_OSM	MRS_O	ICE	SM_TO	MRS_RE
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 498. CTUEFR field descriptions

Field	Description
ET_OE	External Trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
T1_OE	Timer 1 trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.

Table 498. CTUEFR field descriptions (continued)

Field	Description
T0_OE	Timer 0 trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
ADC_OE	ADC command generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
TGS_OSM	TGS Overrun in Sequential Mode 0 Error has not occurred. 1 Error has occurred.
MRS_O	Master Reload Signal Overrun 0 Error has not occurred. 1 Error has occurred.
ICE	Invalid Command Error 0 Error has not occurred. 1 Error has occurred.
SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode 0 Error has not occurred. 1 Error has occurred.
MRS_RE	Master Reload Signal Reload Error 0 Error has not occurred. 1 Error has occurred.

26.8.18 Cross triggering unit interrupt flag register (CTUIFR)

Figure 517. Cross triggering unit interrupt flag register (CTUIFR)

Address: Base + 0x00C2 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	ADC_I	T7_I	T6_I	T5_I	T4_I	T3_I	T2_I	T1_I	T0_I	MRS_I
W							r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 499. CTUIFR field descriptions

Field	Description
ADC_I	ADC command interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T7_I	Trigger 7 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T6_I	Trigger 6 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 499. CTUIFR field descriptions (continued)

Field	Description
T5_I	Trigger 5 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T4_I	Trigger 4 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T3_I	Trigger 3 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T2_I	Trigger 2 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T1_I	Trigger 1 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T0_I	Trigger 0 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
MRS_I	MRS Interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

26.8.19 Cross triggering unit interrupt/DMA register (CTUIR)

Figure 518. Cross triggering unit interrupt/DMA register (CTUIR)

Address: Base + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_IE	T6_IE	T5_IE	T4_IE	T3_IE	T2_IE	T1_IE	T0_IE	0	0	0	0	0	MRS_DMAE	MRS_IE	IEE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 500. CTUIR field descriptions

Field	Description
T7_IE	Trigger 7 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T6_IE	Trigger 6 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled

Table 500. CTUIR field descriptions (continued)

Field	Description
T5_IE	Trigger 5 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T4_IE	Trigger 4 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T3_IE	Trigger 3 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T2_IE	Trigger 2 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T1_IE	Trigger 1 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T0_IE	Trigger 0 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
MRS_DMAE	DMA transfer Enable on MRS occurrence if GRE bit is set 0 Interrupt disabled 1 Interrupt enabled
MRS_IE	MRS Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
IEE	Interrupt Error Enable 0 Interrupt disabled 1 Interrupt enabled

26.8.20 Control ON time register (COTR)

Figure 519. Control ON time register (COTR)

Address: Base + 0x00C6 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	COTR							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 501. COTR field descriptions

Field	Description
COTR	Control ON-Time and Guard Time for external trigger

26.8.21 Cross triggering unit control register (CTUCR)

Figure 520. Cross triggering unit control register (CTUCR)

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_SG	T6_SG	T5_SG	T4_SG	T3_SG	T2_SG	T1_SG	T0_SG	CTU_ADC_R	CTU_ODIS	DFE	CGRE	FGRE	MRS_SG	GRE	TGSISR_RE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 502. CTUCR field descriptions

Field	Description
T7_SG	Trigger 7 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T6_SG	Trigger 6 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T5_SG	Trigger 5 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T4_SG	Trigger 4 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T3_SG	Trigger 3 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T2_SG	Trigger 2 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T1_SG	Trigger 1 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T0_SG	Trigger 0 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
CTU_ADC_R	CTU/ADC state machine Reset
CTU_ODIS	CTU Output Disable
DFE	Digital Filter Enable This bit can be read by software and can only be set to 1 by software.
CGRE	Clear GRE This bit can only be set to 1 by software.

Table 502. CTUCR field descriptions (continued)

Field	Description
FGRE	Flag GRE
MRS_SG	MRS Software Generated
GRE	General Reload Enable This bit can be read by software and can only be set to 1 by software.
TGSISR_RE	TGS Input Selection Register Reload Enable This bit can be read by software and can only be set to 1 by software.

26.8.22 Cross triggering unit digital filter (CTUDF)

Figure 521. Cross triggering unit digital filter (CTUDF)

Address: Base + 0x00CA Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	N							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 503. CTUDF field descriptions

Field	Description
0-7	Reserved
8-15 N	Digital Filter value (the external signal is considered at 1 if it is latched N time at 1 and is considered at 0 if it is latched N time at 0)

26.8.23 Cross triggering unit power control register (CTUPCR)

Figure 522. Cross triggering unit power control register (CTUPCR)

Address: Base + 0x00CC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 504. CTUPCR field descriptions

Field	Description
0-14	Reserved
15 MDIS	Module Disable

27 Semaphore Unit (SEMA4)

27.1 Introduction

SPC56xP60/54 contains two SEMA4 units.

In a dual-processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The semaphore unit (SEMA4) provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

In this chapter, the two instantiations of the e200z0h core on SPC56xP60/54 are referred to as e200z0h_0 and e200z0h_1. (The e200z0h_0 instantiation runs from reset in dual processor mode.)

27.1.1 Block diagram

Figure 523 is a simplified block diagram of the SEMA4 unit that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

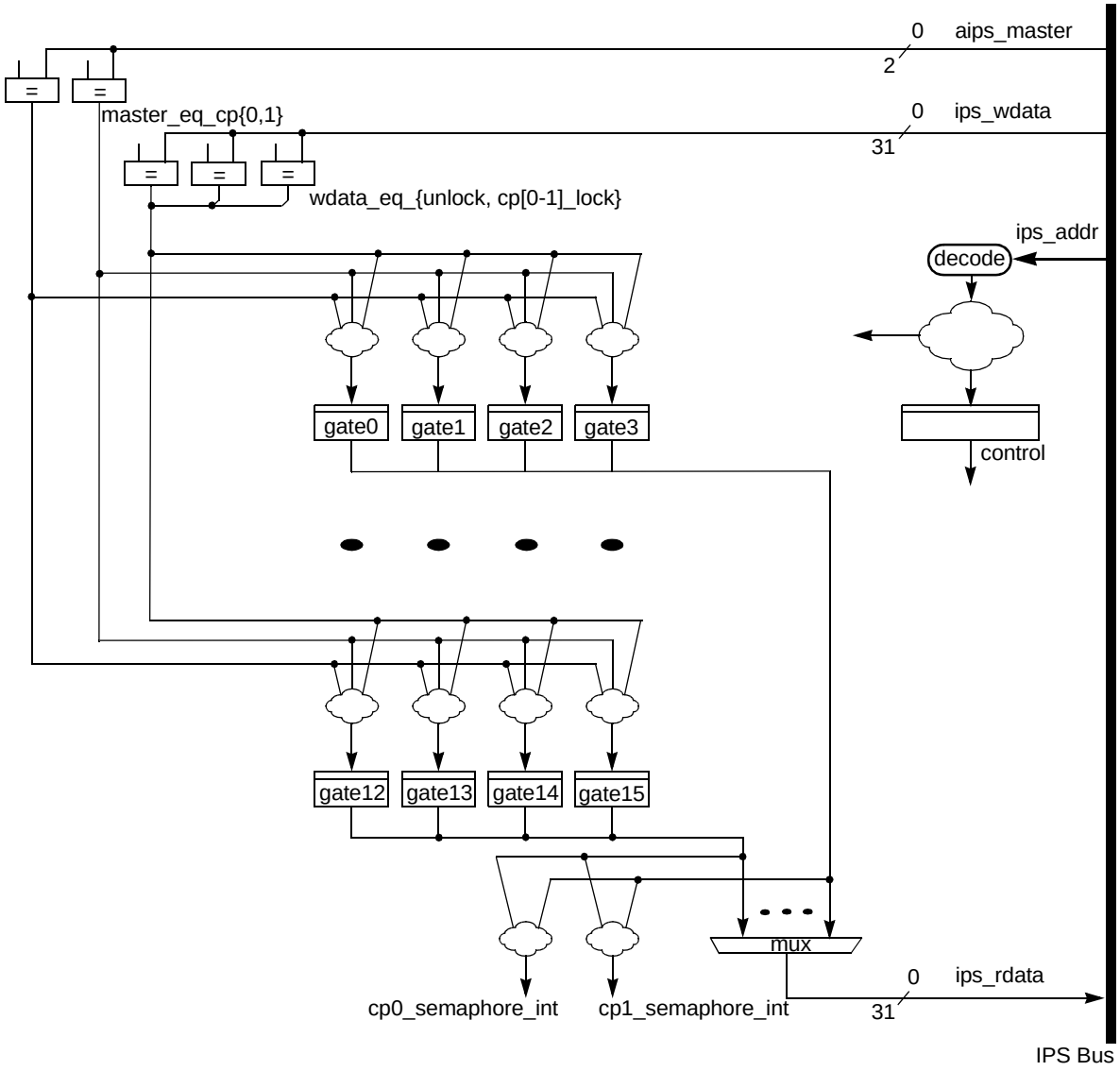


Figure 523. SEMA4 block diagram

27.1.2 Features

The SEMA4 unit implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration
- Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes
  - Three-state implementation
  - if gate = 0b00, then state = unlocked
  - if gate = 0b01, then state = locked by e200z0h_0 (master ID = 0)
  - if gate = 0b10, then state = locked by e200z0h_1 (master ID = 1)
  - Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
  - After it is locked, the gate must be unlocked by a write of zeroes from the locking processor
- Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked
- Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability

Note: Semaphore gates that are locked when entering sleep mode are cleared by the internal reset generated when exiting sleep mode.

27.1.3 Modes of operation

The SEMA4 unit does not support any special modes of operation.

27.2 Signal description

The SEMA4 unit does not include any external signals.

27.3 Memory map and register description

The SEMA4 programming model map is shown in *Table 505*. The address of each register is given as an offset to the SEMA4 base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 505. SEMA4 memory map

Offset from SEMA4_BASE 0xFFF2_4000 (SEMA4_0) 0x8FF2_4000 (SEMA4_1)	Register	Location
0x0000	SEMA4_Gate00—Semaphores gate 0	*on page 884*
0x0001	SEMA4_Gate01—Semaphores gate 1	*on page 884*
0x0002	SEMA4_Gate02—Semaphores gate 2	*on page 884*
0x0003	SEMA4_Gate03—Semaphores gate 3	*on page 884*

Table 505. SEMA4 memory map (continued)

Offset from SEMA4_BASE 0xFFF2_4000 (SEMA4_0) 0x8FF2_4000 (SEMA4_1)	Register	Location
0x0004	SEMA4_Gate04—Semaphores gate 4	on page 884
0x0005	SEMA4_Gate05—Semaphores gate 5	on page 884
0x0006	SEMA4_Gate06—Semaphores gate 6	on page 884
0x0007	SEMA4_Gate07—Semaphores gate 7	on page 884
0x0008	SEMA4_Gate08—Semaphores gate 8	on page 884
0x0009	SEMA4_Gate09—Semaphores gate 9	on page 884
0x000A	SEMA4_Gate10—Semaphores gate 10	on page 884
0x000B	SEMA4_Gate11—Semaphores gate 11	on page 884
0x000C	SEMA4_Gate12—Semaphores gate 12	on page 884
0x000D	SEMA4_Gate13—Semaphores gate 13	on page 884
0x000E	SEMA4_Gate14—Semaphores gate 14	on page 884
0x000F	SEMA4_Gate15—Semaphores gate 15	on page 884
0x0010–0x003F	Reserved	
00x040	SEMA4_CP0INE—Semaphores CP0 IRQ notification enable	on page 885
0x0042–0x0047	Reserved	
0x0048	SEMA4_CP1INE—Semaphores CP1 IRQ notification enable	on page 885
0x004A–0x07F	Reserved	
0x0080	SEMA4_CP0NTF—Semaphores CP0 IRQ notification	on page 886
0x0082–00x087	Reserved	
0x0088	SEMA4_CP1NTF—Semaphores CP1 IRQ notification	on page 885
0x008A–0x00FF	Reserved	
0x0100	SEMA4_RSTGT—Semaphores reset gate	on page 886
0x0102–0x0103	Reserved	
0x0104	SEMA4_RSTNTF—Semaphores reset IRQ notification	on page 888
0x0106–0x3FFF	Reserved	

27.3.1 Semaphores gate *n* register (SEMA4_GATE*n*)

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the

write data operand must update the state of a single gate only. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Offset: *n* (*n* = 0x0, 0x1,0x2,..., 0xF) Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	GTFSM	
W								
Reset	0	0	0	0	0	0	0	0

Figure 524. SEMA4 gate *n* register (SEMA4_GATE*n*)

Table 506. SEMA4_GATE*n* field descriptions

Field	Description
GTFSM	Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as: 00 The gate is unlocked (free). 01 The gate has been locked by processor 0. 10 The gate has been locked by processor 1. 11 This state encoding is never used and therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine. *Note: The state of the gate reflects the last processor that locked it, which can be useful during system debug.*

27.3.2 Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly-enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP*n*INE) and the interrupt request register (SEMA4_CP*n*NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP*n*INE registers, unimplemented bits read as zeroes and writes are ignored.

Offset: 0x0040 (SEMA4_CP0INE)
0x0048 (SEMA4_CP1INE) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INE0	INE1	INE2	INE3	INE4	INE5	INE6	INE7	INE8	INE9	INE10	INE11	INE12	INE13	INE14	INE15
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 525. Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE)

Table 507. SEMA4_CP{0,1}NTF field descriptions

Field	Description
INE*n*	Interrupt Request Notification Enable *n*. This field is a bitmap to enable the generation of an interrupt notification from a failed attempt to lock gate *n*. 0 The generation of the notification interrupt is disabled. 1 The generation of the notification interrupt is enabled.

27.3.3 Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

- When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
- After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
- When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN*n*] and SEMA4_CP1NTF[GN*n*].

Offset: 0x0080 (SEMA4_CP0NTF)
0x0088 (SEMA4_CP1NTF)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 526. Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF)

Table 508. SEMA4_CP{0,1}NTF field descriptions

Field	Description
GN*n*	Gate *n* Notification. This read-only field is a bitmap of the interrupt request notification from a failed attempt to lock gate *n*. 0 No notification interrupt generated. 1 Notification interrupt generated.

27.3.4 Semaphores (secure) reset gate *n* (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the SEMA4 unit implements a secure reset mechanism which allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the

servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xE2; the least significant byte is a “don’t care” for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) which implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]) and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: 0x0100 (SEMA4_RSTGT) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTGSM		0	RSTGMS			RSTGTN							
W	RSTGDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 527. Semaphores (secure) reset gate *n* (SEMA4_RSTGT)

Table 509. SEMA4_RSTGT field descriptions

Field	Description
RSTGSM	Reset Gate Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. 11 This state encoding is never used and therefore reserved. Reads of the SEMA4_RSTGT register return the encoded state machine value. Note the RSTGSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.

Table 509. SEMA4_RSTGT field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RSTGMS</td><td>Reset Gate Bus Master. This 3-bit read-only field records the logical number of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.

<table>
<tr><th>Master</th><th>Master ID</th></tr>
<tr><td>e200z0h_0</td><td>0</td></tr>
<tr><td>e200z0h_1</td><td>1</td></tr>
<tr><td>eDMA</td><td>2</td></tr>
<tr><td>FlexRay</td><td>3</td></tr>
<tr><td>—</td><td>4</td></tr>
<tr><td>—</td><td>5</td></tr>
<tr><td>—</td><td>6</td></tr>
<tr><td>—</td><td>7</td></tr>
</table>
</td></tr>
<tr><td>RSTGTN</td><td>Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write.
If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.</td></tr>
<tr><td>RSTGDP</td><td>Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xe2 while the second write requires RSTGDP = 0x1d.</td></tr>
</table>

27.3.5 Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, it is recognized that system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the SEMA4 unit implements a secure reset mechanism which allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure reset mechanism requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a “don’t care” for this reference.
2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN])

specifies the notification(s) to be reset. This field can specify a single notification be cleared or that all notifications are cleared.

3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTNSM		0	RSTNMS			RSTNTN							
W	RSTNDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 528. Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF)

Table 510. SEMA4_RSTNTF field descriptions

Field	Description
RSTNSM	Reset Notification Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The two-write sequence has completed. Generate the specified notification reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. 11 This state encoding is never used and therefore reserved. Reads of the SEMA4_RSTNTF register return the encoded state machine value. Note the RSTNSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.
RSTNMS	Reset Notification Bus Master. This 3-bit read-only field records the logical number of the bus master performing the notification reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs. **Master** / **Master ID** e200z0h_0 / 0 e200z0h_1 / 1 eDMA / 2 FlexRay / 3 — / 4 — / 5 — / 6 — / 7

Table 510. SEMA4_RSTNTF field descriptions (continued)

Field	Description
RSTNTN	Reset Notification Number. This 8-bit field specifies the specific IRQ notification state machine to be reset. This field is updated by the second write. If RSTNTN < 64, then reset the single IRQ notification machine defined by RSTNTN, else reset all the notifications.
RSTNDP	Reset Notification Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the notification reset mechanism. For the first write, RSTNDP = 0x47 while the second write requires RSTNDP = 0xb8.

27.4 Functional description

Multi-processor systems require a function that can be used to safely and easily provide a locking mechanism that is then used by system software to control access to shared data structures, shared hardware resources, and etc. These gating mechanisms are used by the software to serialize (and synchronize) writes to shared data and/or resources to prevent race conditions and preserve memory coherency between processes and processors.

For example, if processor X enters a section of code where shared data values are to be updated or read coherently, it must first acquire a semaphore. This locks, or closes, a software gate. After the gate has been locked, a properly architected software system does not allow other processes (or processors) to execute the same code segment or modify the shared data structure protected by the gate, that is, other processes/processors are locked out. Many software implementations include a spin-wait loop within the lock function until the locking of the gate is accomplished. After the lock has been obtained, processor X continues execution and updates the data values protected by the particular lock. After the updates are complete, processor X unlocks (or opens) the software gate, allowing other processes/processors access to the updated data values.

There are three important rules that must be followed for a correctly implemented system solution:

- All writes to shared data values or shared hardware resources must be protected by a gate variable.
- After a processor locks a gate, accesses to the shared data or resources by other processes/processors must be blocked. This is enforced by software conventions.
- The processor that locks a particular gate is the only processor that can unlock, or open, that gate.

Information in the hardware gate identifying the locking processor can be useful for system-level debugging.

The Hennessy/Patterson text on computer architecture offers this description of software gating:

> “One of the major requirements of a shared-memory architecture multiprocessor is being able to coordinate processes that are working on a common task. Typically, a programmer will use *lock variables* to synchronize the processes.
>
> The difficulty for the architect of a multiprocessor is to provide a mechanism to decide which processor gets the lock and to provide the operation that locks a variable. Arbitration is easy for shared-bus multiprocessors, since the bus is the only path to memory. The processor that gets the bus locks out all the other processors from memory. If the CPU and bus provide an atomic swap operation, programmers can

create locks with the proper semantics. The adjective *atomic* is key, for it means that a processor can both read a location **and** set it to the locked value in the same bus operation, preventing any other processor from reading or writing memory." [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 471-472]

The classic text continues with a description of the steps required to lock/unlock a variable using an atomic swap instruction.

"Assume that 0 means unlocked and 1 means locked. A processor first reads the lock variable to test its state. A processor keeps reading and testing until the value indicates that the lock is unlocked. The processor then races against all other processes that were similarly "spin waiting" to see who can lock the variable first. All processes use a swap instruction that reads the old value and stores a 1 into the lock variable. The single winner will see the 0, and the losers will see a 1 that was placed there by the winner. (The losers will continue to set the variable to the locked value, but that doesn't matter.) The winning processor executes the code after the lock and then stores a 0 into the lock when it exits, starting the race all over again. Testing the old value and then setting to a new value is why the atomic swap instruction is called *test and set* in some instruction sets." [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 472-473]

The sole drawback to a hardware-based semaphore module is the limited number of semaphores versus the infinite number that can be supported with Power Architecture reservation instructions.

27.4.1 Semaphore usage

Example 1: Inter-processor communication done with software interrupts and semaphores...

- The e200z0h_1 uses software interrupts to tell the e200z0h_0 that new data is available, or the e200z0h_0 does the same to tell the e200z0h_1 that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIU to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM not being updated, then the semaphore must first be locked, then the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
  - For example, if both cores want to update the same location, then the following sequence may occur.

    1. The e200z0h_1 locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to the e200z0h_0.

    2. Before the e200z0h_0 takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.

    3. The e200z0h_0 software interrupt ISR reads the data sent to the e200z0h_1, not the data sent from the e200z0h_1, and performs an incorrect operation.
  - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores...

- The e200z0h_0 wants to coherently read a section of shared memory.
- The e200z0h_0 should check that the semaphore for the shared memory is not currently set.
- The e200z0h_0 should set the semaphore for the shared memory to prevent the e200z0h_1 from updating the shared memory.
- The e200z0h_0 reads the required data, then unlock the semaphore.

27.5 Initialization information

The reset state of the SEMA4 unit allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the unit to immediately begin operation.

27.6 Application information

In an operational multi-core system, most interactions involving the SEMA4 unit involves reads and writes to the SEMA4_GATE*n* registers for implementation of the hardware-enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate
  - The processor performs a byte write of logical_processor_number + 1 to gate[i]
  - The processor reads back gate[i] and checks for a value of logical_processor_number + 1

    If the compare indicates the expected value

    then the gate is locked; proceed with the protected code segment

    else

    lock operation failed;

    repeat process beginning with byte write to gate[i] in spin-wait loop, or

    proceed with another execution path and wait for failed lock interrupt notification

A simple C-language example of a gatelock function is shown in *Example 15*. This function follows the Hennessy/Patterson example.

Example 15 Sample Gatelock Function

```
#define UNLOCK      0
#define CP0_LOCK    1
#define CP1_LOCK    2

void gateLock (n)
int   n;             /* gate number to lock */
{
  int   i;
  int   current_value;
  int   locked_value;

  i = processor_number();  /* obtain logical CPU number */

  if (i == 0)
```

```
    locked_value = CP0_LOCK;
  else
    locked_value = CP1_LOCK;

  /* read the current value of the gate and wait until the state == UNLOCK */
  do {
    current_value = gate[n];
  } while (current_value != UNLOCK);

  /* the current value of the gate == UNLOCK. attempt to lock the gate for this
     processor. spin-wait in this loop until gate ownership is obtained */
  do {
    gate[n] = locked_value;   /* write gate with processor_number + 1 */
   current_value = gate[n];  /* read gate to verify ownership was obtained */
  } while (current_value != locked_value);
}
```

- To unlock (open) a gate
  - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate

In this example, a reference to `processor_number()` is used to retrieve this hardware configuration value. Typically, the logical processor numbers are defined by a hardwired input vector to the individual cores. For PowerPC cores, there is a processor ID register (PIR) which is SPR 286 and contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: mfspr rx,286 where rx is the destination GPRn. Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., **`rdcpn rx`** for a read CPU number instruction.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CP*n*INE, SEMA4_ CP*n*NTF) are required. There is no required negation of the failed lock write notification interrupt as the request is automatically negated by the SEMA4 unit once the gate has been successfully locked by the failing processor.

Finally, in the event a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable(s) (SEMA4_CP*n*INE) bits be disabled before initiating the secure reset 2-write sequence to avoid any race conditions involving spurious notification interrupt requests.

27.7 DMA requests

There are no DMA requests associated with the SEMA4 unit.

28 eTimer

28.1 Introduction

The eTimer module contains six identical counter/timer channels and one watchdog timer function only for eTimer_0. Each 16-bit counter/timer channel contains a prescaler, a counter, a load register, a hold register, two queued capture registers, two compare registers, two compare preload registers, and four control registers.

The Load register provides the initialization value to the counter when the counter's terminal value has been reached. For true modulo counting the counter can also be initialized by the CMPLD1 or CMPLD2 registers.

The Hold register captures the counter's value when other counters are being read. This feature supports the reading of cascaded counters coherently.

The Capture registers enable an external signal to take a "snapshot" of the counter's current value.

The COMP1 and COMP2 registers provide the values to which the counter is compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled.

The Prescaler provides different time bases useful for clocking the counter/timer.

The Counter provides the ability to count internal or external events.

Within the eTimer module (set of six timer/counter channels) the input pins are shareable.

28.2 Features

The eTimer module design includes these features:

- Six 16-bit counters/timers
- Count up/down
- Cascadeable counters
- Enhanced programmable up/down modulo counting
- Max count rate equals peripheral clock/2 for external clocks
- Max count rate equals peripheral clock for internal clocks
- Count once or repeatedly
- Preloadable counters
- Preloadable compare registers
- Counters can share available input pins
- Separate prescaler for each counter
- Each counter has capture and compare capability
- Continuous and single shot capture for enhanced speed measurement
- DMA support of capture registers and compare registers
- 32-bit watchdog capability to detect stalled quadrature counting (implemented only on eTimer_0)
- OFLAG comparison for safety critical applications
- Programmable operation during debug mode and stop mode
- Programmable input filter
- Counting start can be synchronized across counters

28.3 Module block diagram

The eTimer block diagram is shown in *Figure 529*.

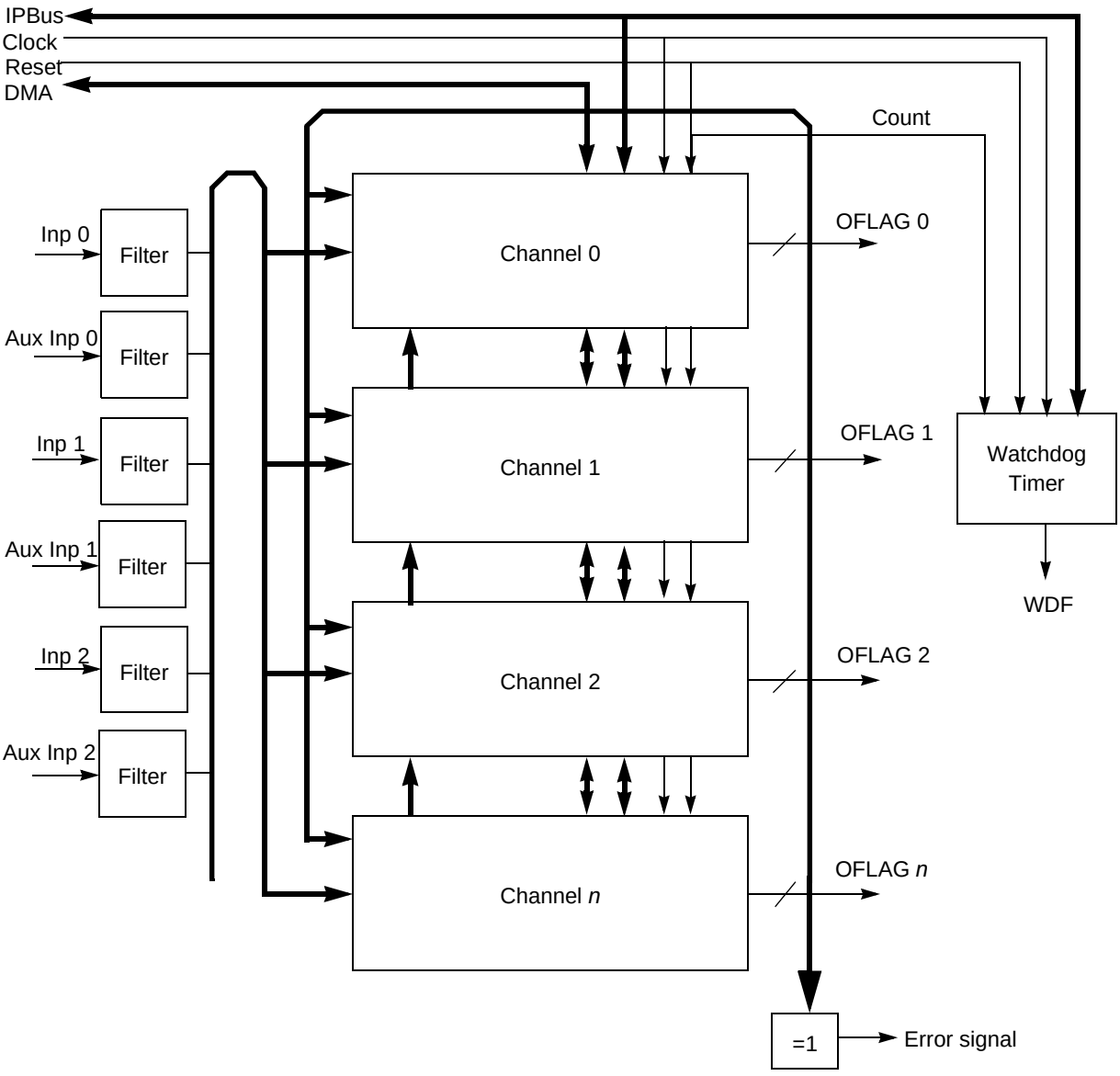


Figure 529. eTimer block diagram

28.4 Channel block diagram

Each of the timer/counter channels within the eTimer are shown in *Figure 530*.

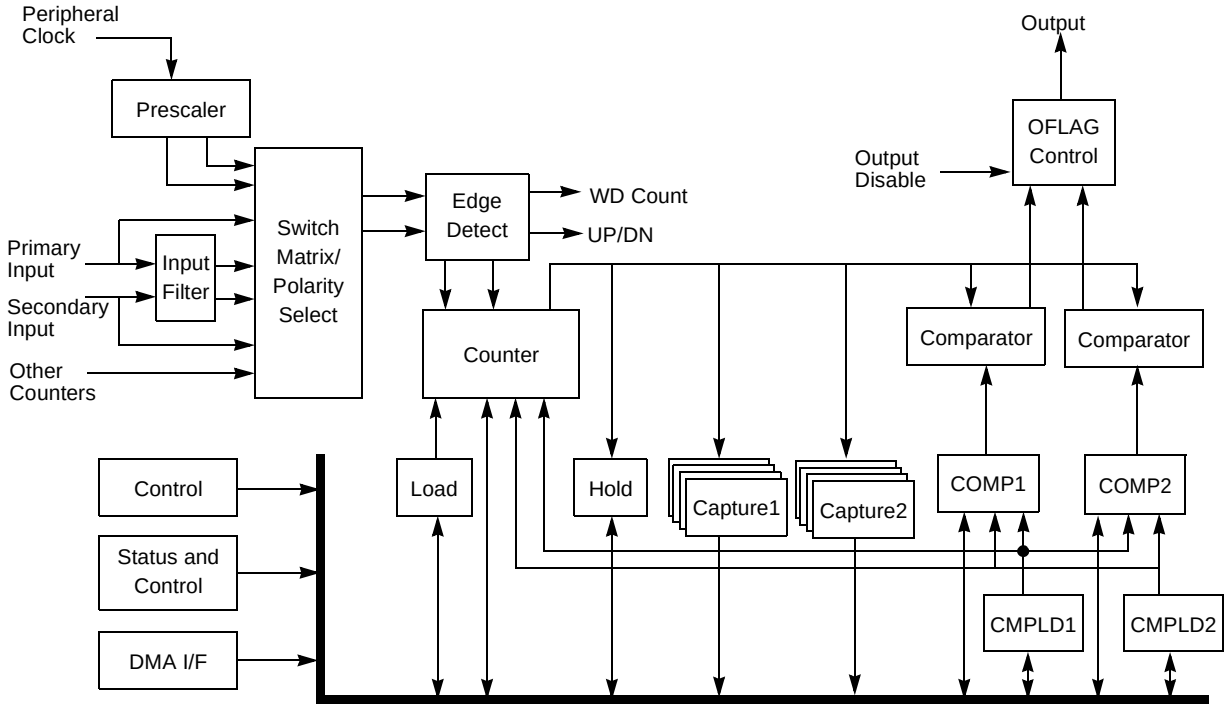


Figure 530. eTimer channel block diagram

28.5 External signal descriptions

The eTimer module has six external signals that can be used as either inputs or outputs.

28.5.1 ETC[5:0]—eTimer input/outputs

These pins can be independently configured to be either timer input sources or output flags.

28.6 Memory map and registers

28.6.1 Overview

Table 511 shows the memory map for the eTimer modules.

Table 511. eTimer memory map

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
eTimer Channel 0		
0x0000	COMP1—Compare Register 1	*on page 901*
0x0002	COMP2—Compare Register 2	*on page 902*
0x0004	CAPT1—Capture Register 1	*on page 902*
0x0006	CAPT2—Capture Register 2	*on page 903*
0x0008	LOAD—Load Register	*on page 903*
0x000A	HOLD—Hold Register	*on page 904*
0x000C	CNTR—Counter Register	*on page 904*
0x000E	CTRL1—Control Register 1	*on page 905*
0x0010	CTRL2—Control Register 2	*on page 907*
0x0012	CTRL3—Control Register 3	*on page 909*
0x0014	STS—Status Register	*on page 910*
0x0016	INTDMA—Interrupt and DMA Enable Register	*on page 912*
0x0018	CMPLD1—Comparator Load Register 1	*on page 913*
0x001A	CMPLD2—Comparator Load Register 2	*on page 913*
0x001C	CCCTRL—Compare and Capture Control Register	*on page 914*
0x001E	FILT—Input Filter Register	*on page 916*
eTimer Channel 1		
0x0020	COMP1—Compare Register 1	*on page 901*
0x0022	COMP2—Compare Register 2	*on page 902*
0x0024	CAPT1—Capture Register 1	*on page 902*
0x0026	CAPT2—Capture Register 2	*on page 903*
0x0028	LOAD—Load Register	*on page 903*
0x002A	HOLD—Hold Register	*on page 904*
0x002C	CNTR—Counter Register	*on page 904*
0x002E	CTRL1—Control Register 1	*on page 905*
0x0030	CTRL2—Control Register 2	*on page 907*
0x0032	CTRL3—Control Register 3	*on page 909*
0x0034	STS—Status Register	*on page 910*
0x0036	INTDMA—Interrupt and DMA Enable Register	*on page 912*
0x0038	CMPLD1—Comparator Load Register 1	*on page 913*
0x003A	CMPLD2—Comparator Load Register 2	*on page 913*

Table 511. eTimer memory map (continued)

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
0x003C	CCCTRL—Compare and Capture Control Register	on page 914
0x003E	FILT—Input Filter Register	on page 916
eTimer Channel 2		
0x0040	COMP1—Compare Register 1	on page 901
0x0042	COMP2—Compare Register 2	on page 902
0x0044	CAPT1—Capture Register 1	on page 902
0x0046	CAPT2—Capture Register 2	on page 903
0x0048	LOAD—Load Register	on page 903
0x004A	HOLD—Hold Register	on page 904
0x004C	CNTR—Counter Register	on page 904
0x004E	CTRL1—Control Register 1	on page 905
0x0050	CTRL2—Control Register 2	on page 907
0x0052	CTRL3—Control Register 3	on page 909
0x0054	STS—Status Register	on page 910
0x0056	INTDMA—Interrupt and DMA Enable Register	on page 912
0x0058	CMPLD1—Comparator Load Register 1	on page 913
0x005A	CMPLD2—Comparator Load Register 2	on page 913
0x005C	CCCTRL—Compare and Capture Control Register	on page 914
0x005E	FILT—Input Filter Register	on page 916
eTimer Channel 3		
0x0060	COMP1—Compare Register 1	on page 901
0x0062	COMP2—Compare Register 2	on page 902
0x0064	CAPT1—Capture Register 1	on page 902
0x0066	CAPT2—Capture Register 2	on page 903
0x00068	LOAD—Load Register	on page 903
0x006A	HOLD—Hold Register	on page 904
0x006C	CNTR—Counter Register	on page 904
0x006E	CTRL1—Control Register 1	on page 905
0x0070	CTRL2—Control Register 2	on page 907
0x0072	CTRL3—Control Register 3	on page 909
0x0074	STS—Status Register	on page 910
0x0076	INTDMA—Interrupt and DMA Enable Register	on page 912

Table 511. eTimer memory map (continued)

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
0x0078	CMPLD1—Comparator Load Register 1	on page 913
0x007A	CMPLD2—Comparator Load Register 2	on page 913
0x007C	CCCTRL—Compare and Capture Control Register	on page 914
0x007E	FILT—Input Filter Register	on page 916
eTimer Channel 4		
0x0080	COMP1—Compare Register 1	on page 901
0x0082	COMP2—Compare Register 2	on page 902
0x0084	CAPT1—Capture Register 1	on page 902
0x0086	CAPT2—Capture Register 2	on page 903
0x0088	LOAD—Load Register	on page 903
0x008A	HOLD—Hold Register	on page 904
0x008C	CNTR—Counter Register	on page 904
0x008E	CTRL1—Control Register 1	on page 905
0x0090	CTRL2—Control Register 2	on page 907
0x0092	CTRL3—Control Register 3	on page 909
0x0094	STS—Status Register	on page 910
0x0096	INTDMA—Interrupt and DMA Enable Register	on page 912
0x0098	CMPLD1—Comparator Load Register 1	on page 913
0x009A	CMPLD2—Comparator Load Register 2	on page 913
0x009C	CCCTRL—Compare and Capture Control Register	on page 914
0x009E	FILT—Input Filter Register	on page 916
eTimer Channel 5		
0x00A0	COMP1—Compare Register 1	on page 901
0x00A2	COMP2—Compare Register 2	on page 902
0x00A4	CAPT1—Capture Register 1	on page 902
0x00A6	CAPT2—Capture Register 2	on page 903
0x00A8	LOAD—Load Register	on page 903
0x00AA	HOLD—Hold Register	on page 904
0x00AC	CNTR—Counter Register	on page 904
0x00AE	CTRL1—Control Register 1	on page 905
0x00B0	CTRL2—Control Register 2	on page 907
0x00B2	CTRL3—Control Register 3	on page 909

Table 511. eTimer memory map (continued)

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
0x00B4	STS—Status Register	*on page 910*
0x00B6	INTDMA—Interrupt and DMA Enable Register	*on page 912*
0x00B8	CMPLD1—Comparator Load Register 1	*on page 913*
0x00BA	CMPLD2—Comparator Load Register 2	*on page 913*
0x00BC	CCCTRL—Compare and Capture Control Register	*on page 914*
0x00BE	FILT—Input Filter Register	*on page 916*
0x00C0–0x00FF	Reserved	
0x0100	WDTOL—Watchdog Time-out Low Register	*on page 917*
0x0102	WDTOH—Watchdog Time-out High Register	*on page 917*
0x0104–0x010B	Reserved	
Watchdog and Configuration registers		
0x010C	ENBL—Channel Enable Register	*on page 917*
0x0110	DREQ0—DMA Request 0 Select Register	*on page 918*
0x0112	DREQ1—DMA Request 1 Select Register	*on page 918*
0x0114–0x3FFF	Reserved	

28.6.2 Timer channel registers

These registers are repeated for each timer channel. The base address of channel 0 is the same as the base address of the eTimer module as a whole. The base address of channel 1 is 0x20. This is the base address of the eTimer module plus an offset based on the number of bytes of registers in a timer channel. The base address of each subsequent timer channel is equal to the base address of the previous channel plus this same offset of 0x20.

Compare register 1 (COMP1)

The COMP1 register stores the value used for comparison with the counter value. More explanation on the use of COMP1 can be found in *Section : Usage of compare registers*.

Figure 531. Compare register 1 (COMP1)

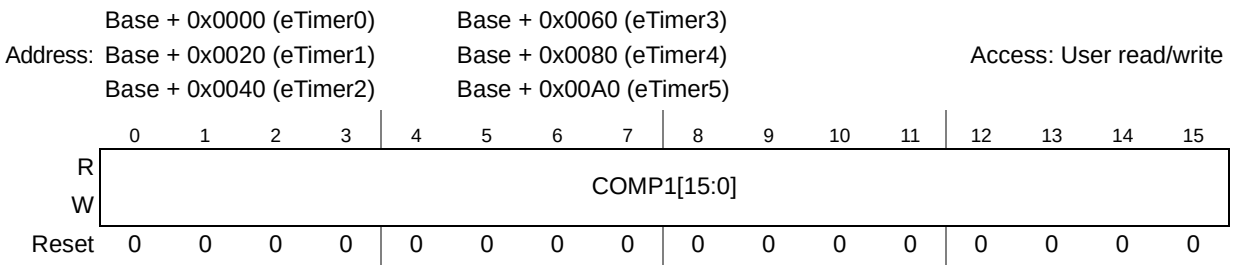


Table 512. COMP1 field descriptions

Field	Description
COMP1[15:0]	Compare 1 Stores the value used for comparison with the counter value. This register is not byte accessible.

Compare register 2 (COMP2)

The COMP2 register stores the value used for comparison with the counter value. More explanation on the use of COMP2 can be found in *Section : Usage of compare registers.*

Figure 532. Compare register 2 (COMP2)

Address: Base + 0x0002 (eTimer0), Base + 0x0022 (eTimer1), Base + 0x0042 (eTimer2), Base + 0x0062 (eTimer3), Base + 0x0082 (eTimer4), Base + 0x00A2 (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	COMP2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 513. COMP2 field descriptions

Field	Description
COMP2[15:0]	Compare 2 Stores the value used for comparison with the counter value. This register is not byte accessible.

Capture register 1 (CAPT1)

The CAPT1 register stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT1MODE bits in the Compare and Capture Control (CCCTRL) register. This is actually a -deep FIFO and not a single register.

Figure 533. Capture register 1 (CAPT1)

Address: Base + 0x0004 (eTimer0), Base + 0x0024 (eTimer1), Base + 0x0044 (eTimer2), Base + 0x0064 (eTimer3), Base + 0x0084 (eTimer4), Base + 0x00A4 (eTimer5)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPT1[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 514. CAPT1 field descriptions

Field	Description
CAPT1[15:0]	Capture 1 Stores the value captured from the counter. This register is not byte accessible.

Capture register 2 (CAPT2)

This read only register stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT2MODE bits. This is actually a 2-deep FIFO and not a single register. This register is not byte accessible.

Figure 534. Capture register 2 (CAPT2)

Address: Base + 0x0006 (eTimer0), Base + 0x0026 (eTimer1), Base + 0x0046 (eTimer2), Base + 0x0066 (eTimer3), Base + 0x0086 (eTimer4), Base + 0x00A6 (eTimer5)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPT2[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 515. CAPT2 field descriptions

Field	Description
CAPT2[15:0]	Capture 2 Stores the value captured from the counter. This register is not byte accessible.

Load register (LOAD)

This read/write register stores the value used to initialize the counter. This register is not byte accessible.

Figure 535. Load register (LOAD)

Address: Base + 0x0008 (eTimer0), Base + 0x0028(eTimer1), Base + 0x0048 (eTimer2), Base + 0x0068 (eTimer3), Base + 0x0088 (eTimer4), Base + 0x00A8 (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	LOAD[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 516. LOAD field descriptions

Field	Description
LOAD[15:0]	Load Stores the value used to initialize the counter. This register is not byte accessible.

Hold register (HOLD)

This read-only register stores the counter's value whenever any of the other counters within a module are read. This supports coherent reading of cascaded counters.

Figure 536. Hold register (HOLD)

Address: Base + 0x000A (eTimer0), Base + 0x002A (eTimer1), Base + 0x004A (eTimer2), Base + 0x006A (eTimer3), Base + 0x008A (eTimer4), Base + 0x00AA (eTimer5) — Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HOLD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 517. HOLD field descriptions

Field	Description
HOLD[15:0]	Stores the counter's value whenever any of the other counters within a module are read. The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

Counter register (CNTR)

This read/write register is the counter for this channel of the timer module. This register is not byte accessible.

Figure 537. Counter register (CNTR)

Address: Base + 0x000C (eTimer0), Base + 0x002C (eTimer1), Base + 0x004C (eTimer2), Base + 0x006C (eTimer3), Base + 0x008C (eTimer4), Base + 0x00AC (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTR[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 518. CNTR field descriptions

Field	Description
CNTR[15:0]	Contains the count value for this channel of the eTimer module. This register is not byte accessible.

Control register 1 (CTRL1)

Figure 538. Control register 1 (CTRL1)

Address: Base + 0x000E(eTimer0), Base + 0x002E (eTimer1), Base + 0x004E (eTimer2), Base + 0x006E (eTimer3), Base + 0x008E (eTimer4), Base + 0x00AE (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTMODE[2:0]				PRISRC[4:0]				ONCE	LENGTH	DIR		SECSRC[4:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 519. CTRL1 field descriptions

Field	Description
CNTMODE[2:0]	Count Mode These bits control the basic counting and behavior of the counter. 000 No Operation. 001 Count rising edges of primary source. Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1. If primary count source is IP bus clock, only rising edges are counted regardless of PIPS value. 010 Count rising and falling edges of primary source. IP Bus clock divide by 1 can not be used as a primary count source in edge count mode. 011 Count rising edges of primary source while secondary input high active. 100 Quadrature count mode, uses primary and secondary sources. 101 Count primary source rising edges, secondary source specifies direction (1 = minus). Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1. 110 Edge of secondary source triggers primary count till compare. 111 Cascaded counter mode, up/down. Primary count source must be set to one of the counter outputs.
PRISRC	Primary Count Source These bits select the primary count source. See *Table 520*. A timer cannot select its own output as its primary count source. If this is done, the timer will not count.
ONCE	Count Once This bit selects continuous or one-shot counting mode. 0 Count repeatedly. 1 Count until compare and then stop. When output mode 0x4 is used, the counter reinitializes after reaching the COMP1 value and continues to count to the COMP2 value, then stops.

Table 519. CTRL1 field descriptions (continued)

Field	Description
LENGTH	Count Length This bit determines whether the counter counts to the compare value and then reinitializes itself to the value specified in the LOAD, CMPLD1, or CMPLD2 registers, or the counter continues counting past the compare value, to the binary roll over. 0 Continue counting to roll over. 1 Count until compare, then reinitialize. The value that the counter is reinitialized with depends on the settings of CLC1 and CLC2. If neither of these indicates the counter is to be loaded from one of the CMPLD registers, then the LOAD register reinitializes the counter upon matching either COMP register. If one of CLC1 or CLC2 indicates that the counter is to be loaded from one of the CMPLD registers, then the counter will reinitialize to the value in the appropriate CMPLD register upon a match with the appropriate COMP register. If both of the CLC1 and CLC2 fields indicate that the counter is to be loaded from the CMPLD registers, then CMPLD1 will have priority if both compares happen at the same value. When output mode 0x4 is used, alternating values of COMP1 and COMP2 are used to generate successful comparisons. For example, the counter counts until COMP1 value is reached, reinitializes, then counts until COMP2 value is reached, reinitializes, then counts until COMP1 value is reached, etc.
DIR	Count Direction This bit selects either the normal count direction up, or the reverse direction, down. 0Count up. 1Count down.
SECSRC	Secondary Count Source These bits identify the source to be used as a count command or timer command. The selected input can trigger the timer to capture the current value of the CNTR register. The selected input can also be used to specify the count direction. The polarity of the signal can be inverted by the SIPS bit of the CTRL2 register.

Table 520. Count source values

Value	Meaning	Value	Meaning
00000	Counter #0 input pin	10000	Counter #0 output
00001	Counter #1 input pin	10001	Counter #1 output
00010	Counter #2 input pin	10010	Counter #2 output
00011	Counter #3 input pin	10011	Counter #3 output
00100	Counter #4 input pin	10100	Counter #4 output
00101	Counter #5 input pin	10101	Counter #5 output
00110	*Reserved*	10110	*Reserved*
00111	*Reserved*	10111	*Reserved*
01000	Auxiliary input #0 pin	11000	IP Bus clock divide by 1 prescaler
01001	Auxiliary input #1 pin	11001	IP Bus clock divide by 2 prescaler
01010	Auxiliary input #2 pin	11010	IP Bus clock divide by 4 prescaler
01011	*Reserved*	11011	IP Bus clock divide by 8 prescaler

Table 520. Count source values (continued)

Value	Meaning	Value	Meaning
01100	*Reserved*	11100	IP Bus clock divide by 16 prescaler
01101	*Reserved*	11101	IP Bus clock divide by 32 prescaler
01110	*Reserved*	11110	IP Bus clock divide by 64 prescaler
01111	*Reserved*	11111	IP Bus clock divide by 128 prescaler

Control register 2 (CTRL2)

Figure 539. Control register 2 (CTRL2)

Address: Base + 0x0010 (eTimer0), Base + 0x0030 (eTimer1), Base + 0x0050 (eTimer2), Base + 0x0070 (eTimer3), Base + 0x0090 (eTimer4), Base + 0x00B0 (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OEN	RDNT	INPUT	VAL	0	COFRC	COINIT		SIPS	PIPS	OPS	MSTR	OUTMODE[3:0]			
W					FORCE											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 521. CTRL2 field descriptions

Field	Description
OEN	Output Enable This bit determines the direction of the external pin. 0 The external pin is configured as an input. 1 OFLAG output signal is driven on the external pin. Other timer channels using this external pin as their input will see the driven value. The polarity of the signal will be determined by the OPS bit.
RDNT	Redundant Channel Enable This bit enables redundant channel checking between adjacent channels (0 and 1, 2 and 3, 4 and 5). When this bit is cleared, the RCF bit in this channel cannot be set. When this bit is set, the RCF bit is set by a miscompare between the OFLAG of this channel and the OFLAG of its redundant adjacent channel, which causes the output of this channel to go inactive (logic 0 prior to consideration of the OPS bit). 0Disable redundant channel checking. 1Enable redundant channel checking.
INPUT	External input signal This read only bit reflects the current state of the signal selected via SECSRC after application of the SIPS bit and filtering.
VAL	Forced OFLAG Value This bit determines the value of the OFLAG output signal when a software triggered FORCE command occurs.

Table 521. CTRL2 field descriptions (continued)

Field	Description
FORCE	Force the OFLAG output This write only bit forces the current value of the VAL bit to be written to the OFLAG output. This bit always reads as a zero. The VAL and FORCE bits can be written simultaneously in a single write operation. Write to the FORCE bit only if OUTMODE is 0000 (software controlled). Setting this bit while the OUTMODE is a different value may yield unpredictable results.
COFRC	Co-channel OFLAG Force This bit enables the compare from another channel within the module to force the state of this counter's OFLAG output signal. 0 Other channels cannot force the OFLAG of this channel. 1 Other channels may force the OFLAG of this channel.
COINIT	Co-channel Initialization These bits enable another channel within the module to force the reinitialization of this channel when the other channel has an active compare event. 00 Other channels cannot force reinitialization of this channel. 01 Other channels may force a reinitialization of this channel's counter using the LOAD reg. 10 Other channels may force a reinitialization of this channel's counter with the CMPLD2 reg when this channel is counting down or the CMPLD1 reg when this channel is counting up. 11 Reserved.
SIPS	Secondary Source Input Polarity Select This bit inverts the polarity of the signal selected by the SECSRC bits. 0 True polarity. 1 Inverted polarity.
PIPS	Primary Source Input Polarity Select This bit inverts the polarity of the signal selected by the PRISRC bits. This only applies if the signal selected by PRISRC is not the prescaled IP Bus clock. 0 True polarity. 1 Inverted polarity.
OPS	Output Polarity Select This bit inverts the OFLAG output signal polarity. 0 True polarity. 1 Inverted polarity.

Table 521. CTRL2 field descriptions (continued)

Field	Description
MSTR	Master Mode This bit enables the compare function's output to be broadcast to the other channels in the module. The compare signal then can be used to reinitialize the other counters and/or force their OFLAG signal outputs. 0 Disable broadcast of compare events from this channel. 1 Enable broadcast of compare events from this channel.
OUTMODE	Output Mode These bits determine the mode of operation for the OFLAG output signal. 0000 Software controlled 0001 Clear OFLAG output on successful compare (COMP1 or COMP2) 0010 Set OFLAG output on successful compare (COMP1 or COMP2) 0011 Toggle OFLAG output on successful compare (COMP1 or COMP2) 0100 Toggle OFLAG output using alternating compare registers 0101 Set on compare with COMP1, cleared on secondary source input edge 0110 Set on compare with COMP2, cleared on secondary source input edge 0111 Set on compare, cleared on counter roll-over 1000 Set on successful compare on COMP1, clear on successful compare on COMP2 1001 Asserted while counter is active, cleared when counter is stopped. 1010 Asserted when counting up, cleared when counting down. 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Enable gated clock output while counter is active

Control register 3 (CTRL3)

Figure 540. Control register 3 (CTRL3)

Address: Base + 0x0012 (eTimer0), Base + 0x0032 (eTimer1), Base + 0x0052 (eTimer2), Base + 0x0072 (eTimer3), Base + 0x0092 (eTimer4), Base + 0x00B2 (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STPEN	ROC		0	1	1	1	1	C2FCNT[2:0]			C1FCNT[2:0]			DBGEN [1:0]	
W																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Table 522. CTRL3 field descriptions

Field	Description
STPEN	Stop Actions Enable This bit allows the tristating of the timer output during stop mode. 0 Output enable is unaffected by stop mode. 1 Output enable is disabled during stop mode.
ROC	Reload on Capture These bits enable the capture function to cause the counter to be reloaded from the LOAD register. 00 Do not reload the counter on a capture event. 01 Reload the counter on a capture 1 event. 10 Reload the counter on a capture 2 event. 11 Reload the counter on both a capture 1 event and a capture 2 event.
C2FCNT	CAPT2 FIFO Word Count This field reflects the number of words in the CAPT2 FIFO.
C1FCNT	CAPT1 FIFO Word Count This field reflects the number of words in the CAPT1 FIFO.
DBGEN	Debug Actions Enable These bits allow the counter channel to perform certain actions in response to the device entering debug mode. 00 Continue with normal operation during debug mode. (default) 01 Halt channel counter during debug mode. 10 Force OFLAG to logic 0 (prior to consideration of the OPS bit) during debug mode. 11 Both halt counter and force OFLAG to 0 during debug mode.

Status register (STS)

Figure 541. Status register (STS)

Address: Base + 0x0014 (eTimer0), Base + 0x0034 (eTimer1), Base + 0x0054 (eTimer2), Base + 0x0074 (eTimer3), Base + 0x0094 (eTimer4), Base + 0x00B4 (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	WDF	RCF	ICF2	ICF1	IEHF	IELF	TOF	TCF2	TCF1	TCF
W							w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 523. STS field descriptions

Field	Description
WDF	Watchdog Time-out Flag This bit is set when the watchdog times out by counting down to zero. The watchdog must be enabled for time-out to occur and channel 0 must be in quadrature decode count mode (CNTMODE = 100). This bit is cleared by writing a 1 to this bit. This bit is used in channel 0 only.
RCF	Redundant Channel Flag This bit is set when there is a miscompare between this channel's OFLAG value and the OFLAG value of the corresponding redundant channel. Corresponding channels are grouped together in the following pairs: 0 and 1, 2 and 3, 4 and 5, or 6 and 7. This bit can only be set if the RDNT bit is set. This bit is cleared by writing a 1 to this bit. This bit is used in even channels (0, 2, 4, and 6) only.
ICF2	Input Capture 2 Flag This bit is set when an input capture event (as defined by CPT2MODE) occurs while the counter is enabled and the word count of the CAPT2 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a 1 to this bit if ICF2DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF2DE is set (DMA).
ICF1	Input Capture 1 Flag This bit is set when an input capture event (as defined by CPT1MODE) occurs while the counter is enabled and the word count of the CAPT1 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a 1 to this bit if ICF1DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF1DE is set (DMA).
IEHF	Input Edge High Flag This bit is set when a positive input transition occurs (on an input selected by SECSRC) while the counter is enabled. This bit is cleared by writing a 1 to this bit.
IELF	Input Edge Low Flag This bit is set when a negative input transition occurs (on an input selected by SECSRC) while the counter is enabled. This bit is cleared by writing a 1 to this bit.
TOF	Timer Overflow Flag This bit is set when the counter rolls over its maximum value 0xFFFF or 0x0000 (depending on count direction). This bit is cleared by writing a 1 to this bit.
TCF2	Timer Compare 2 Flag This bit is set when a successful compare occurs with COMP2. This bit is cleared by writing a 1 to this bit.
TCF1	Timer Compare 1 Flag This bit is set when a successful compare occurs with COMP1. This bit is cleared by writing a 1 to this bit.
TCF	Timer Compare Flag This bit is set when a successful compare occurs. This bit is cleared by writing a 1 to this bit.

Interrupt and DMA enable register (INTDMA)

Figure 542. Interrupt and DMA enable register (INTDMA)

Address: Base + 0x0016 (eTimer0), Base + 0x0036 (eTimer1), Base + 0x0056 (eTimer2), Base + 0x0076 (eTimer3), Base + 0x0096 (eTimer4), Base + 0x00B6 (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ICF2DE	ICF1DE	CMPLD2DE	CMPLD1DE	0	0	WDFIE	RCFIE	ICF2IE	ICF1IE	IEHFIE	IELFIE	TOFIE	TCF2IE	TCF1IE	TCFIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 524. INTDMA field descriptions

Field	Description
ICF2DE	Input Capture 2 Flag DMA Enable Setting this bit enables DMA read requests for CAPT2 when the ICF2 bit is set. Do not set both this bit and the ICF2IE bit.
ICF1DE	Input Capture 1 Flag DMA Enable Setting this bit enables DMA read requests for CAPT1 when the ICF1 bit is set. Do not set both this bit and the ICF1IE bit.
CMPLD2DE	Comparator Load Register 2 Flag DMA Enable Setting this bit enables DMA write requests to the CMPLD2 register whenever data is transferred out of the CMPLD2 reg into either the CNTR, COMP1, or COMP2 registers.
CMPLD1DE	Comparator Load Register 1 Flag DMA Enable Setting this bit enables DMA write requests to the CMPLD1 register whenever data is transferred out of the CMPLD1 reg into either the CNTR, COMP1, or COMP2 registers.
WDFIE	Watchdog Flag Interrupt Enable Setting this bit enables interrupts when the WDF bit is set. This bit is used in channel 0 only.
RCFIE	Redundant Channel Flag Interrupt Enable Setting this bit enables interrupts when the RCF bit is set. This bit is used in even channels (0, 2, 4) only.
ICF2IE	Input Capture 2 Flag Interrupt Enable Setting this bit enables interrupts when the ICF2 bit is set. Do not set both this bit and the ICF2DE bit.
ICF1IE	Input Capture 1 Flag Interrupt Enable Setting this bit enables interrupts when the ICF1 bit is set. Do not set both this bit and the ICF1DE bit.
IEHFIE	Input Edge High Flag Interrupt Enable Setting this bit enables interrupts when the IEHF bit is set.
IELFIE	Input Edge Low Flag Interrupt Enable Setting this bit enables interrupts when the IELF bit is set.
TOFIE	Timer Overflow Flag Interrupt Enable Setting this bit enables interrupts when the TOF bit is set.

Table 524. INTDMA field descriptions (continued)

Field	Description
TCF2IE	Timer Compare 2 Flag Interrupt Enable Setting this bit enables interrupts when the TCF2 bit is set.
TCF1IE	Timer Compare 1 Flag Interrupt Enable Setting this bit enables interrupts when the TCF1 bit is set.
TCFIE	Timer Compare Flag Interrupt Enable Setting this bit enables interrupts when the TCF bit is set.

Comparator Load register 1 (CMPLD1)

This read/write register is the preload value for the COMP1 register. This register can also be used to load into the CNTR register. This register is not byte accessible. More information on the use of this register can be found in *Section : Usage of Compare Load registers*.

Figure 543. Comparator Load 1 (CMPLD1)

Address: Base + 0x0018 (eTimer0), Base + 0x0038 (eTimer1), Base + 0x0058 (eTimer2), Base + 0x0078 (eTimer3), Base + 0x0098 (eTimer4), Base + 0x00B8 (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPLD1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 525. CMPLD1 field descriptions

Field	Description
CMPLD1[15:0]	Specifies the preload value for the COMP1 register.

Comparator Load register 2 (CMPLD2)

This read/write register is the preload value for the COMP2 register. This register can also be used to load into the CNTR register. This register is not byte accessible. More information on the use of this register can be found in *Section : Usage of Compare Load registers*.

Figure 544. Comparator Load 2 (CMPLD2)

Address: Base + 0x001A (eTimer0), Base + 0x003A (eTimer1), Base + 0x005A (eTimer2), Base + 0x007A (eTimer3), Base + 0x009A (eTimer4), Base + 0x00BA (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPLD2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 526. CMPLD2 field descriptions

Field	Description
CMPLD2[15:0]	Specifies the preload value for the COMP2 register.

Compare and Capture Control register (CCCTRL)

Figure 545. Compare and Capture Control register (CCCTRL)

Address: Base + 0x001C (eTimer0), Base + 0x003C (eTimer1), Base + 0x005C (eTimer2), Base + 0x007C (eTimer3), Base + 0x009C (eTimer4), Base + 0x00BC (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CLC2[1:0]				CLC1[1:0]		CMPMODE [1:0]		CPT2 MODE[1:0]		CPT1 MODE[1:0]		CFWM[1:0]		ONE SHOT	ARM
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 527. CCCTRL field descriptions

Field	Description
CLC2	Compare Load Control 2 These bits control when COMP2 is preloaded. It also controls the loading of CNTR. 000 Never preload. 001 Reserved 010 Load COMP2 with CMPLD1 upon successful compare with the value in COMP1. 011 Load COMP2 with CMPLD1 upon successful compare with the value in COMP2. 100 Load COMP2 with CMPLD2 upon successful compare with the value in COMP1. 101 Load COMP2 with CMPLD2 upon successful compare with the value in COMP2. 110 Load CNTR with CMPLD2 upon successful compare with the value in COMP1. 111 Load CNTR with CMPLD2 upon successful compare with the value in COMP2.
CLC1	Compare Load Control 1 These bits control when COMP1 is preloaded. It also controls the loading of CNTR. 000 Never preload. 001 Reserved 010 Load COMP1 with CMPLD1 upon successful compare with the value in COMP1. 011 Load COMP1 with CMPLD1 upon successful compare with the value in COMP2. 100 Load COMP1 with CMPLD2 upon successful compare with the value in COMP1. 101 Load COMP1 with CMPLD2 upon successful compare with the value in COMP2. 110 Load CNTR with CMPLD1 upon successful compare with the value in COMP1. 111 Load CNTR with CMPLD1 upon successful compare with the value in COMP2.
CMPMODE	Compare Mode These bits control when the COMP1 and COMP2 registers are used in regards to the counting direction. 00 COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting up. 01 COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting up. 10 COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting down. 11 COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting down.

Table 527. CCCTRL field descriptions (continued)

Field	Description
CPT2MODE	Capture 2 Mode Control These bits control the operation of the CAPT2 register as well as the operation of the ICF2 flag by defining which input edges cause a capture event. The input source is the secondary count source. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
CPT1MODE	Capture 1 Mode Control These bits control the operation of the CAPT1 register as well as the operation of the ICF1 flag by defining which input edges cause a capture event. The input source is the secondary count source. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
CFWM	Capture FIFO Water Mark This field represents the water mark level for the CAPT1 and CAPT2 FIFOs. The capture flags, ICF1 and ICF2, are not set until the word count of the corresponding FIFO is greater than this water mark level.
ONESHOT	One-Shot Capture Mode This bit selects between free-running and one-shot mode for the input capture circuitry. If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and the ARM bit is cleared. No further captures will be performed until the ARM bit is set again. If only one of the capture circuits is enabled, then a single capture will occur on the enabled capture circuit and the ARM bit is then cleared. If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and capture circuit 1 is re-armed. The process continues indefinitely. If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit. 0 Free-running mode is selected 1 One-shot mode is selected.
ARM	Arm Capture Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self cleared when in one-shot mode and the enabled capture circuit(s) has had a capture event(s). 0 Input capture operation is disabled. 1 Input capture operation as specified by the CPT1MODE and CPT2MODE bits is enabled.

Input Filter Register (FILT)

Figure 546. Input Filter register (FILT)

Address: Base + 0x001E (eTimer0), Base + 0x003E (eTimer1), Base + 0x005E (eTimer2), Base + 0x007E (eTimer3), Base + 0x009E (eTimer4), Base + 0x00BE (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FILT_CNT[2:0]			FILT_PER[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 528. FILT field descriptions

Field	Description
FILT_CNT[2:0]	Input Filter Sample Count These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in *Section : Input filter considerations*.
FILT_PER[7:0]	Input Filter Sample Period These bits represent the sampling period (in IPBus clock cycles) of the eTimer input signal. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is 0x00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in *Section : Input filter considerations*.

Input filter considerations

The FILT_PER value should be set such that the sampling period is larger the period of the expected noise. This way a noise spike will only corrupt one sample. The FILT_CNT value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT + 3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of: $\{[(\text{FILT\_CNT} + 3) \times \text{FILT\_PER}] + 2\}$ IPBus clock periods.

28.6.3 Watchdog timer registers

The base address of the Watchdog Timer registers is equal to the base address of the eTimer plus an offset of 0x100. These registers are implemented only on eTimer_0.

Watchdog Time-Out registers (WDTOL and WDTOH)

Figure 547. Watchdog Time-out Low Word register (WDTOL)

Address: Base + 0x0100 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDTOL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 548. Watchdog Time-Out High Word register (WDTOH)

Address: Base + 0x0102 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDTOH															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 529. WDTOL, WDTOH field descriptions

Field	Description
WDTO	Watchdog Time-out These registers are combined to form the 32-bit time-out count for the Timer watchdog function. This time-out count is used to monitor for inactivity on the inputs when channel 0 is in the quadrature decode count mode. The watchdog function is enabled whenever WDTO contains a non-zero value (although actual counting only occurs if channel 0 is in quadrature decode counting mode). The watchdog time-out down counter is loaded whenever WDTOH is written. These registers are not byte accessible. See *Section : Watchdog timer* for more information on the use of the watchdog timer. Note: The Watchdog registers are implemented only on eTimer_0.

28.6.4 Configuration registers

The base address of the configuration registers is equal to the base address of the eTimer plus an offset of 0x010C.

Channel Enable register (ENBL)

Figure 549. Channel Enable register (ENBL)

Address: Base + 0x010C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	ENBL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Table 530. ENBL field descriptions

Field	Description
ENBL	Timer Channel Enable These bits enable the prescaler (if it is being used) and counter in each channel. Multiple ENBL bits can be set at the same time to synchronize the start of separate channels. If an ENBL bit is set, then the corresponding channel will start counting as soon as the CNTMODE field has a value other than 000. When an ENBL bit is clear, the corresponding channel maintains its current value. 0 Timer channel is disabled. 1 Timer channel is enabled. (default)

DMA Request Select registers (DREQ0, DREQ1)

Figure 550. DMA Request 0 Select register (DREQ0)

Address: Base + 0x0110 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DREQ0_EN	0	0	0	0	0	0	0	0	0	0	DREQ0[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 551. DMA Request 1 Select register (DREQ1)

Address: Base + 0x0112 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DREQ1_EN	0	0	0	0	0	0	0	0	0	0	DREQ1[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 531. DREQ*n* field descriptions

Field	Description
DREQ*n*_EN	DMA Request Enable Use these bits to enable each of the four module level DMA request outputs. Program the DREQ fields prior to setting the corresponding enable bit. Clearing this enable bit will remove the request but wíll not clear the flag that is causing the request. 1 = DMA request enabled. 0 = DMA request disabled.
DREQ*n*	DMA Request Select Use these fields to select which DMA request source will be muxed onto one of the two module level DMA request outputs. Make sure each of the DREQ registers is programmed with a different value else a single DMA source will cause multiple DMA requests. Enable a DMA request in the channel specific INTDMA register after the DREQ registers are programmed. 00000Channel 0 CAPT1 DMA read request 00001Channel 0 CAPT2 DMA read request 00010 Channel 0 CMPLD1 DMA write request 00011 Channel 0 CMPLD2 DMA write request 00100 Channel 1 CAPT1 DMA read request 00101 Channel 1 CAPT2 DMA read request 00110 Channel 1 CMPLD1 DMA write request 00111 Channel 1 CMPLD2 DMA write request 01000 Channel 2 CAPT1 DMA read request 01001 Channel 2 CAPT2 DMA read request 01010 Channel 2 CMPLD1 DMA write request 01011 Channel 2 CMPLD2 DMA write request 01100 Channel 3 CAPT1 DMA read request 01101 Channel 3 CAPT2 DMA read request 01110 Channel 3 CMPLD1 DMA write request 01111 Channel 3 CMPLD2 DMA write request 10000 Channel 4 CAPT1 DMA read request 10001 Channel 4 CAPT2 DMA read request 10010 Channel 4 CMPLD1 DMA write request 10011 Channel 4 CMPLD2 DMA write request 10100 Channel 5 CAPT1 DMA read request 10101 Channel 5 CAPT2 DMA read request 10110 Channel 5 CMPLD1 DMA write request 10111 Channel 5 CMPLD2 DMA write request

28.7 Functional description

28.7.1 General

Each channel has two basic modes of operation: it can count internal or external events, or it can count an internal clock source while an external input signal is asserted, thus timing the width of the external input signal.

- The counter can count the rising, falling, or both edges of the selected input pin.
- The counter can decode and count quadrature encoded input signals.
- The counter can count up and down using dual inputs in a “count with direction” format.
- The counter’s terminal count value (modulo) is programmable.
  - The value that is loaded into the counter after reaching its terminal count is programmable.
- The counter can count repeatedly, or it can stop after completing one count cycle.
- The counter can be programmed to count to a programmed value and then immediately reinitialize, or it can count through the compare value until the count “rolls over” to zero.

The external inputs to each counter/timer are shareable among each of the channels within the module. The external inputs can be used as:

- Count commands
- Timer commands
- They can trigger the current counter value to be “captured”
- They can be used to generate interrupt requests

The polarity of the external inputs is selectable.

The primary output of each channel is the output signal OFLAG. The OFLAG output signal can be:

- Set, cleared, or toggled when the counter reaches the programmed value.
- The OFLAG output signal may be output to an external pin instead of having that pin serve as a timer input.
- The OFLAG output signal enables each counter to generate square waves, or pulse stream outputs.
- The polarity of the OFLAG output signal is programmable.

Any channel can be assigned as a Master. A master’s compare signal can be broadcast to the other channels within the module. The other channels can be configured to reinitialize their counters and/or force their OFLAG output signals to predetermined values when a Master channel’s compare event occurs.

28.7.2 Counting modes

The selected external signals are sampled at the eTimer’s base clock rate and then run through a transition detector. The maximum count rate is one-half of the eTimer’s base clock rate when using an external signal. Internal clock sources can be used to clock the counters at the eTimer’s base clock rate.

If a counter is programmed to count to a specific value and then stop, the CNTMODE field in the CTRL1 register is cleared when the count terminates.

STOP mode

When the CNTMODE field is set to 000, the counter is inert. No counting will occur. Stop mode will also disable the interrupts caused by input transitions on a selected input pin.

COUNT mode

When the CNTMODE field is set to 001, the counter will count the rising edges of the selected clock source. This mode is useful for generating periodic interrupts for timing purposes, or counting external events such as “widgets” on a conveyor belt passing a sensor. If the selected input is inverted by setting the PIPS bit, then the negative edge of the selected external input signal is counted.

See *Section : CASCADE-COUNT mode* through *Section : PULSE-OUTPUT mode* for additional capabilities of this operating mode.

EDGE-COUNT mode

When the CNTMODE field is set to 010, the counter will count both edges of the selected external clock source. This mode is useful for counting the changes in the external environment such as a simple encoder wheel.

GATED-COUNT mode

When the CNTMODE field is set to 011, the counter will count while the selected secondary input signal is high. This mode is used to time the duration of external events. If the selected input is inverted by setting the PIPS bit, then the counter will count while the selected secondary input is low.

QUADRATURE-COUNT mode

When the CNTMODE field is set to 100, the counter will decode the primary and secondary external inputs as quadrature encoded signals. Quadrature signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quadrature signals are square waves that are 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information.

Figure 552 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

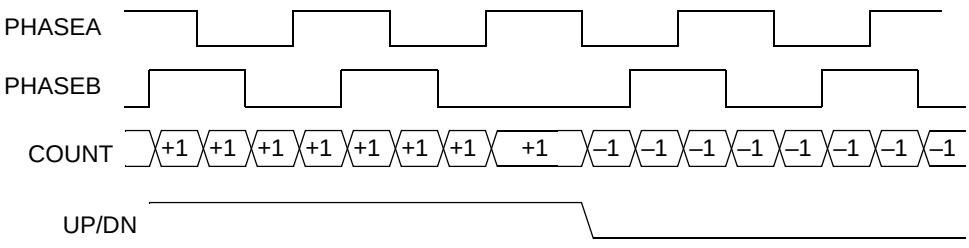


Figure 552. Quadrature incremental position encoder

SIGNED-COUNT mode

When the CNTMODE field is set to 101, the counter counts the primary clock source while the selected secondary source provides the selected count direction (up/down).

TRIGGERED-COUNT mode

When the CNTMODE field is set to 110, the counter will begin counting the primary clock source after a positive transition (negative if SIPS = 1) of the secondary input occurs. The counting will continue until a compare event occurs or another positive input transition is detected. Subsequent secondary positive input transitions will continue to restart and stop the counting until a compare event occurs.

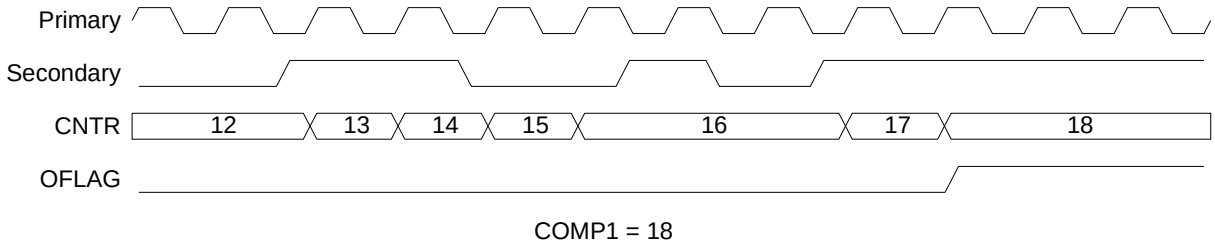


Figure 553. Triggered Count mode (length = 1)

ONE-SHOT mode

When the CNTMODE field is set to 110 and the counter is set to reinitialize at a compare event (LENGTH = 1), and the OFLAG OUTMODE is set to 0101 (cleared on init, set on compare), the counter works in ONE-SHOT mode. If an external events causes the counter to count, when terminal count is reached, the output is asserted. This delayed output can be used to provide timing delays.

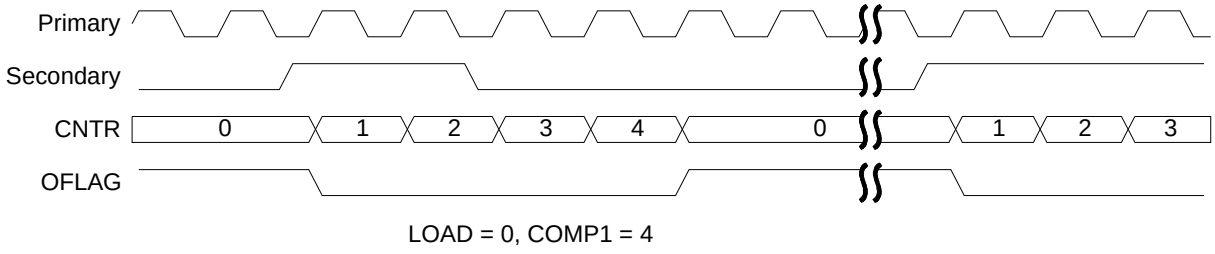


Figure 554. One-Shot mode (length = 1)

CASCADE-COUNT mode

When the CNTMODE field is set to 111, the counter's input is connected to the output of another selected counter. The counter will count up and down as compare events occur in the selected source counter. This cascaded or "daisy-chained" mode enables multiple counters to be cascaded to yield longer counter lengths. When operating in cascade mode, a special high speed signal path is used between modules rather than the OFLAG output signal. If the selected source counter is counting up and it experiences a compare event, the counter will be incremented. If the selected source counter is counting down and it experiences a compare event, the counter will be decremented.

Either one or two counters may be cascaded to create a 32-bit wide synchronous counter.

Whenever any counter is read within a counter module, all of the counters' values within the module are captured in their respective HOLD registers. This action supports the reading of a cascaded counter chain. First read any counter of a cascaded counter chain, then read the HOLD registers of the other counters in the chain. The cascaded counter mode is synchronous.

Note: It is possible to connect counters together by using the other (non-cascade) counter modes and selecting the outputs of other counters as a clock source. In this case, the counters are operating in a "ripple" mode, where higher order counters will transition a clock later than a purely synchronous design.

One channel can be cascaded with any other channel, but channels cannot be cascaded more than two deep. Separate cascades of pairs of channels can be created. For example, channels 0 and 1 can be cascaded, and channels and cascaded separately. However, channels 0, 1, and 5 cannot be cascaded.

PULSE-OUTPUT mode

When the counter is set up with CNTMODE = 001, and the OFLAG OUTMODE is set to 1111 (gated clock output), and the ONCE bit is set, then the counter will output a pulse stream of pulses that has the same frequency of the selected clock source, and the number of output pulses is equal to the compare value minus the init value. This mode is useful for driving step motor systems.

Note: This does not work if the PRISRC is set to 11000.

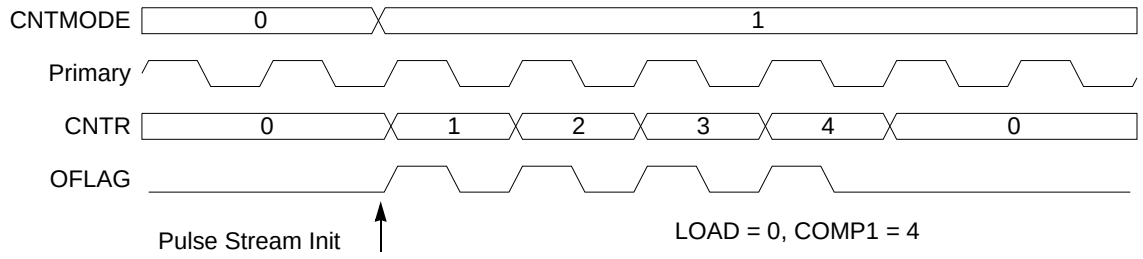


Figure 555. Pulse Output mode

Usage of compare registers

The dual compare registers (COMP1 and COMP2) provide a bidirectional modulo count capability.

The COMP1 register should be set to the desired maximum count value or 0xFFFF to indicate the maximum unsigned value prior to roll-over, and the COMP2 register should be set to the minimum count value or 0x0000 to indicate the minimum unsigned value prior to roll-under.

Use caution when changing COMP1 and COMP2 while the counter is active. If the counter has already passed the new value, it will count to 0xFFFF or 0x0000, roll over, then begin counting toward the new value. The check is: CNTR = COMPx, *not* CNTR > COMP1 or CNTR < COMP2.

Using the CMPLD1 and CMPLD2 registers to preload compare values helps to minimize this problem.

Usage of Compare Load registers

The CMPLD1, CMPLD2, and CCCTRL registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events. To ensure correct functionality while using these registers, the following methods are recommended.

The purpose of the compare load feature is to allow quicker updating of the compare registers. A compare register can be updated using interrupts. However, because of the latency between an interrupt event occurring and the service of that interrupt, there is the possibility that the counter may have already counted past the new compare value by the time the compare register is updated by the interrupt service routine. The counter would then continue counting until it rolled over and reached the new compare value.

To address this, the compare registers are updated in hardware in the same way the counter register is reinitialized to the value stored in the LOAD register. The compare load feature allows the user to calculate new compare values and store them in to the comparator load registers. When a compare event occurs, the new compare values in the comparator load registers are written to the compare registers eliminating the use of software to do this.

MODULO COUNTING mode

To create a modulo counter using COMP1 and COMP2 as the counter boundaries (instead of 0x0000 and 0xFFFF), set the registers in the following manner. Set CNTMODE to either 100 (quadrature count mode) or 101 (count with direction mode). Use count through roll-over (LENGTH = 0) and continuous count (ONCE = 0). Set COMP1 and CMPLD1 to the upper boundary value. Set COMP2 and CMPLD2 to the lower boundary value. Set CMPMODE = 10 (COMP1 is used when counting up and COMP2 is used when counting down). Set CLC2 = 110 (load CNTR with value of CMPLD2 on COMP1 compare) and CLC1 = 111 (load CNTR with value of CMPLD1 on COMP2 compare).

28.7.3 Other features

Redundant OFLAG checking

This mode allows the user to bundle two timer functions generating any pattern to compare their resulting OFLAG behaviors (output signal).

The redundant mode is used to support online checks for functional safety reasons. Whenever a mismatch between the two adjacent channels occurs, it is reported via an interrupt to the core and the two outputs are put into their inactive states. An error is flagged via the RCF flag.

This feature can be tested by forcing a transition on one of the OFLAGs using the VAL and FORCE bits of the channel.

Loopback checking

This mode is always available in that one channel can generate an OFLAG while another channel uses the first channels' OFLAG as its input to be measured and verified to be as expected.

Input capture mode

Input capture measures pulse width (by capturing the counter value on two successive input edges) or waveform period (by capturing the counter value on two consecutive rising edges or two consecutive falling edges). The capture registers store a copy of the counter's value

when an input edge (positive, negative, or both) is detected. The type of edge to be captured by each circuit is determined by the CPT1MODE and CPT2MODE bits whose functionality is shown in *Figure 545*.

The arming logic controls the operation of the capture circuits to allow captures to be performed in a free-running (continuous) or one-shot fashion. In free-running mode, the capture sequences will be performed indefinitely. If both capture circuits are enabled, they will work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one-shot mode, only one capture sequence will be performed. If both capture circuits are enabled, capture circuit 0 is armed first. When a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are capable of generating an interrupt to the CPU.

Master/Slave mode

Any timer channel can be assigned as a Master (MSTR = 1). A Master's compare signal can be broadcast to the other channels within the module. The other counters can be configured to reinitialize their counters (COINIT = 1) and/or force their OFLAG output signals (COFRC = 1) to predetermined values when a Master counter compare event occurs.

Watchdog timer

The watchdog timer monitors for a stalled count when channel 0 is in quadrature count mode. When the watchdog is enabled, it loads the time-out value into a down counter. The down counter counts as long as channel 0 is in quadrature decode count mode. If this down counter reaches 0, an interrupt is asserted. The down counter is reloaded to the time-out value each time the counter value from channel 0 changes. If the channel 0 count value is toggling between two values (indicating a possibly stalled encoder), then the down counter is not reloaded.

28.8 Clocks

The eTimer module implements a protocol clock running at a frequency ≥ 120 MHz.

28.9 Interrupts

Each of the channels within the eTimer can generate an interrupt from several sources. The watchdog also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 532. Interrupt summary

Core Interrupt	Interrupt Flag	Interrupt Enable	Name	Description
eTimer_0				
Channels 0–5	TCF	TCFIE	Compare interrupt	Compare of counter and related compare register
	TCF1	TCF1IE	Compare 1 interrupt	Compare of the counter and COMP1 register
	TCF2	TCF2IE	Compare 2 interrupt	Compare of the counter and COMP2 register
	TOF	TOFIE	Overflow interrupt	Generated on counter roll-over or roll-under
	IELF	IELFIE	Input Low Edge interrupt	Falling edge of the secondary input signal
	IEHF	IEHFIE	Input High Edge interrupt	Rising edge of the secondary input signal
	ICF1	ICF1IE	Input Capture 1 interrupt	Input capture event for CAPT1
	ICF2	ICF2IE	Input Capture 2 interrupt	Input capture event for CAPT2
Watchdog	WDF	WDFIE	Watchdog time-out interrupt	Watchdog has timed out
Redundant Channel Checking	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel
eTimer_1				
Channels 0–5	TCF	TCFIE	Compare interrupt	Compare of counter and related compare register
	TCF1	TCF1IE	Compare 1 interrupt	Compare of the counter and COMP1 register
	TCF2	TCF2IE	Compare 2 interrupt	Compare of the counter and COMP2 register
	TOF	TOFIE	Overflow interrupt	Generated on counter roll-over or roll-under
	IELF	IELFIE	Input Low Edge interrupt	Falling edge of the secondary input signal
	IEHF	IEHFIE	Input High Edge interrupt	Rising edge of the secondary input signal
	ICF1	ICF1IE	Input Capture 1 interrupt	Input capture event for CAPT1
	ICF2	ICF2IE	Input Capture 2 interrupt	Input capture event for CAPT2
Redundant Channel Checking	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel

28.10 DMA

Table 533. DMA summary

DMA Request	DMA Enable	Name	Description
Channels 0–5	ICF1DE	CAPT1 read request	CAPT1 contains a value
	ICF2DE	CAPT2 read request	CAPT2 contains a value
	CMPLD1DE	CMPLD1 write request	CMPLD1 needs an update
	CMPLD2DE	CMPLD2 write request	CMPLD2 needs an update

29 Functional Safety

29.1 Introduction

This chapter describes the following modules that help add reliability to the SPC56xP60/54.

- Register protection module
- Software watchdog timer (SWT)

29.2 Register protection module

29.2.1 Overview

The register protection module offers a mechanism to protect defined memory-mapped address locations in a module under protection from being written. The address locations that can be protected are module-specific.

The register protection module is located between the module under protection and the PBRIDGE. This is shown in *Figure 556*.

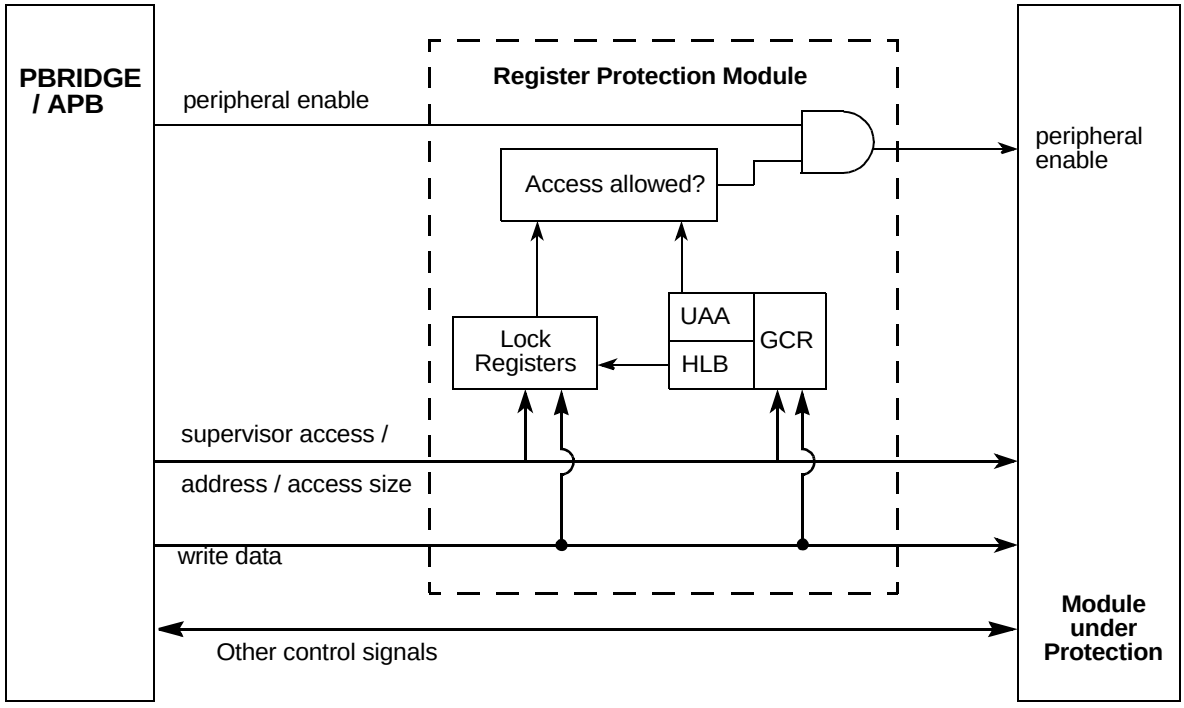


Figure 556. Register protection module block diagram

29.2.2 Features

The register protection module includes these features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock registers for first 6 KB of memory-mapped address space
- Address mirror automatically sets corresponding lock bit
- Once configured lock bits can be protected from changes

29.2.3 Modes of operation

The register protection module is operable when the module under protection is operable.

29.2.4 External signal description

There are no external signals.

29.2.5 Memory map and registers description

This section provides a detailed description of the memory map of a module using the Register protection. The original 16 KB module memory space is divided into five areas as shown in *Figure 557*.

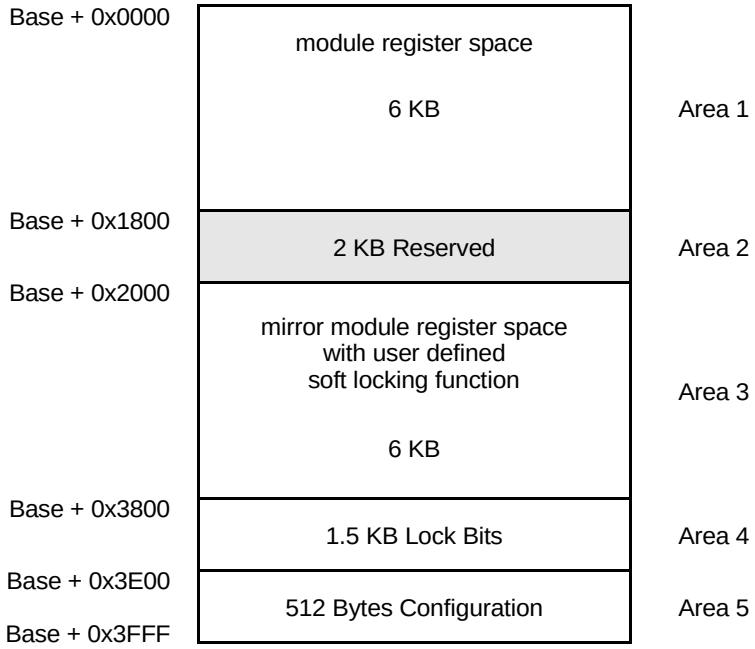


Figure 557. Register protection memory diagram

Area 1 is 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2, 2 KB starting at address 0x1800, is reserved.

Area 3 is 6 KB, starting at address 0x2000 and is a mirror of area 1. A read/write access to these 0x2000 + X addresses will read/write the register at address X. As a side effect, a write access to address 0x2000 + X will set the optional Soft Lock Bits for this address X in the same cycle as the register at address X is written. Not all registers in area 1 need to have protection defined by associated Soft Lock Bits. For unprotected registers at address Y, accesses to address 0x2000 + Y will be identical to accesses at address Y. Only for registers implemented in area 1 and defined as protectable Soft Lock Bits will be available in area 4.

Area 4 is 1.5 KB and holds the Soft Lock Bits, one bit per byte in area 1. The four Soft Lock Bits associated with one module register word are arranged at byte boundaries in the memory map. The Soft Lock Bit registers can be directly written using a bit mask.

Area 5 is 512 bytes and holds the configuration bits of the protection mode. There is one configuration hard lock bit per module that prevents all further modifications to the Soft Lock Bits and can only be cleared by a system reset once set. The other bits, if set, will allow user access to the protected module.

If any locked byte is accessed with a write transaction, a transfer error will be issued to the system and the write transaction will not be executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in areas 4 and 5 will result in a transfer error.

Register protection memory map

Table 534 shows the registers in the Safety Port.

Table 534. Register protection memory map

Offset from REG_ PROT_BASE (0xFFFE_8000)	Register	Location
0x0000–0x17FF	Module Register 0 (MR0)–Module Register 6143(MR6143)	*on page 931*
0x1800–0x1FFF	Reserved	
0x2000–0x37FF	Module Register 0 (MR0) + Set Soft Lock Bit 0 (LMR0)– Module Register 6143 (MR6143) + Set Soft Lock Bit 6143 (LMR6143)	*on page 931*
0x3800–0x3DFF	Soft Lock Bit Register 0 (SLBR0): Soft Lock Bits 0:3– Soft Lock Bit Register 1535 (SLBR1535): Soft Lock Bits 6140:6143	*on page 931*
0x3E00–0x3FFB	Reserved	
0x3FFC	Global Configuration Register (GCR)	*on page 932*

Note: Reserved registers in area #2 will be handled according to the protected IP (module under protection).

Registers description

This section describes in address order all the register protection registers. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

Module registers (MR0–6143)

This is the lower 6 KB module memory space that holds all the functional registers of the module that is protected by the register protection module.

Module Register and Set Soft Lock Bit (LMR0–6143)

This is memory area #3 that provides mirrored access to the MR0–6143 registers with the side effect of setting Soft Lock Bits in case of a write access to a MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in a SLBR*n*[SLB*m*], according to the mapping described in *Table 535*.

Soft Lock Bit Register (SLBR0–1535)

These registers hold the Soft Lock Bits for the protected registers in memory area #1.

Figure 558. Soft Lock Bit Register (SLBR*n*)

Address: Base + 0x3800–0x3DFF Access: User read-only; Supervisor read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	SLB0	SLB1	SLB2	SLB3
W	WE0	WE1	WE2	WE3				
Reset	0	0	0	0	0	0	0	0

Table 535. SLBR*n* field descriptions

Field	Description
WE0 WE1 WE2 WE3	Write Enable Bits for Soft Lock Bits (SLB): WE0 enables writing to SLB0. WE1 enables writing to SLB1. WE2 enables writing to SLB2. WE3 enables writing to SLB3. 0 SLB is not modified. 1 Value is written to SLB.
SLB0 SLB1 SLB2 SLB3	Soft Lock Bits for one MRn register: SLB0 can block accesses to MR[(n × 4) + 0] SLB1 can block accesses to MR[(n × 4) + 1] SLB2 can block accesses to MR[(n × 4) + 2] SLB3 can block accesses to MR[(n × 4) + 3] 0 Associated MRn byte is unprotected and writeable. 1 Associated MRn byte is locked against write accesses.

Table 536 gives some examples how SLBR*n*[SLB] and SLBR*n*[MR*n*] go together:

Table 536. Soft Lock Bits vs. Protected Address

Soft Lock Bit	Protected address
SLBR0[SLB0]	MR0
SLBR0[SLB1]	MR1
SLBR0[SLB2]	MR2
SLBR0[SLB3]	MR3
SLBR1[SLB0]	MR4
SLBR1[SLB1]	MR5
SLBR1[SLB2]	MR6
SLBR1[SLB3]	MR7
SLBR2[SLB0]	MR8
...	...

Global Configuration Register (GCR)

The Global Configuration Register (GCR) controls global configurations related to register protection.

Figure 559. Global Configuration Register (GCR)

Address: Base + 0x3FFC Access: Read Always; Supervisor write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HLB	0	0	0	0	0	0	0	UAA	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 537. GCR field descriptions

Field	Description
HLB	Hard Lock Bit This register cannot be cleared once it is set by software. It can only be cleared by a system reset. 0 All SLB bits are accessible and can be modified. 1 All SLB bits are write protected and can not be modified.
UAA	User Access Allowed 0 The registers in the module under protection can only be written in supervisor mode. All write accesses in non-supervisor mode are not executed and a transfer error is issued. This access restriction is in addition to any access restrictions imposed by the protected IP module. 1 The registers in the module under protection can be accessed in the mode defined for the module registers without any additional restrictions.

Note: The GCR[UAA] bit has no effect on the allowed access modes for the registers in the Register protection module.

29.2.6 Functional description

General

This module provides a generic register (address) write-protection mechanism. The protection size can be:

- 32-bit (address == multiples of 4)
- 16-bit (address == multiples of 2)
- 8-bit (address == multiples of 1)
- unprotected (address == multiples of 1)

For all addresses that are protected there are SLBR*n*[SLB*m*] bits that specify whether the address is locked. When an address is locked it can only be read but not written in any mode (supervisor/normal). If an address is unprotected the corresponding SLBR*n*[SLB*m*] bit is always 0b0 no matter what software is writing to.

Change lock settings

To change the setting whether an address is locked or unlocked, the corresponding SLBR*n*[SLB*m*] bit needs to be changed. This can be done using the following methods:

- Modify the SLBR*n*[SLB*m*] bit directly by writing to area #4
- Set the SLBR*n*[SLB*m*] bit(s) by writing to the mirror module space (area #3)

Both methods are explained in the following sections.

Change lock settings directly via area #4

In memory area #4 the lock bits are located. They can be modified by writing to them. Each SLBR*n*[SLB*m*] bit has a corresponding SLBR*n*[WE*m*] mask bit, which protects it from being modified. This masking makes clear-modify-write operations unnecessary.

Figure 560 shows two modification examples. In the left example there is a write access to the SLBR*n* register specifying a mask value that allows modification of all SLBR*n*[SLB*m*] bits. The example on the right specifies a mask that only allows modification of the bits SLBR*n*[SLB[3:1]].

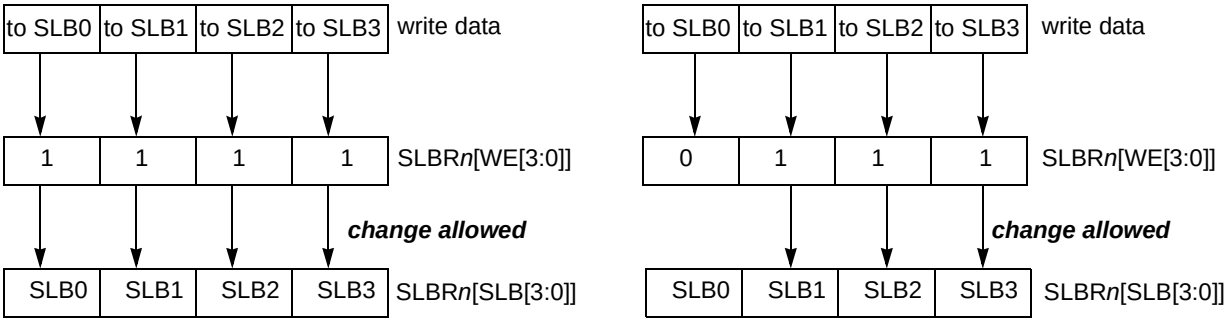


Figure 560. Change lock settings directly via area #4

Figure 560 showed four registers that can be protected 8-bit wise. In *Figure 561* registers with 16-bit protection and in *Figure 562* registers with 32-bit protection are shown.

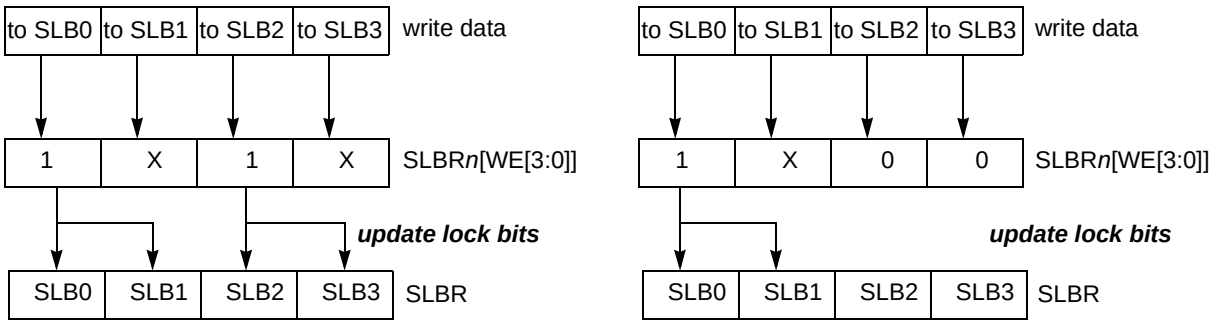


Figure 561. Change lock settings for 16-bit protected addresses

On the right side of *Figure 561* it is shown that the data written to SLBR*n*[SLB0] is automatically written to SLBR*n*[SLB1] also. This is done as the address reflected by SLBR*n*[SLB0] is protected 16-bit wise. Note that in this case the write enable SLBR*n*[WE0] must be set while SLBR*n*[WE1] does not matter. As the enable bits SLBR*n*[WE[3:2]] are cleared the lock bits SLBR*n*[SLB[3:2]] remain unchanged.

In the example on the left side of *Figure 561* the data written to SLBR*n*[SLB0] is mirrored to SLBR*n*[SLB1] and the data written to SLBR*n*[SLB2] is mirrored to SLBR*n*[SLB3] as for both registers the write enables are set.

In *Figure 562* a 32-bit wise protected register is shown. When SLBR*n*[WE0] is set the data written to SLBR*n*[SLB0] is automatically written to SLBR*n*[SLB[3:1]] also. Otherwise SLBR*n*[SLB[3:0]] remains unchanged.

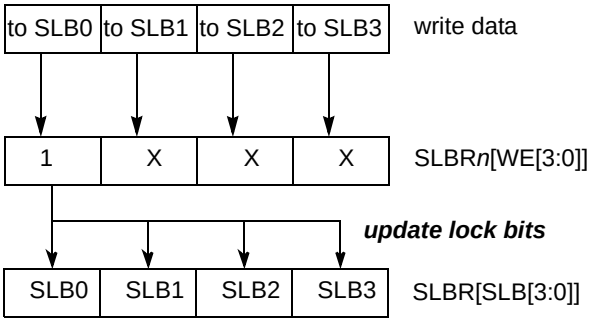


Figure 562. Change lock settings for 32-bit protected addresses

Figure 563 shows an example that has a mixed protection size configuration.

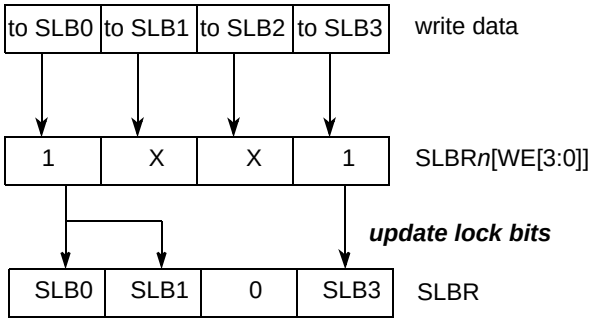


Figure 563. Change lock settings for mixed protection

The data written to SLBR*n*[SLB0] is mirrored to SLBR*n*[SLB1] as the corresponding register is 16-bit protected. The data written to SLBR*n*[SLB2] is blocked as the corresponding register is unprotected. The data written to SLBR*n*[SLB3] is written to SLBR*n*[SLB3].

Enable locking via mirror module space (area #3)

It is possible to enable locking for a register after writing to it. To do so the mirrored module address space must be used. *Figure 564* shows one example.

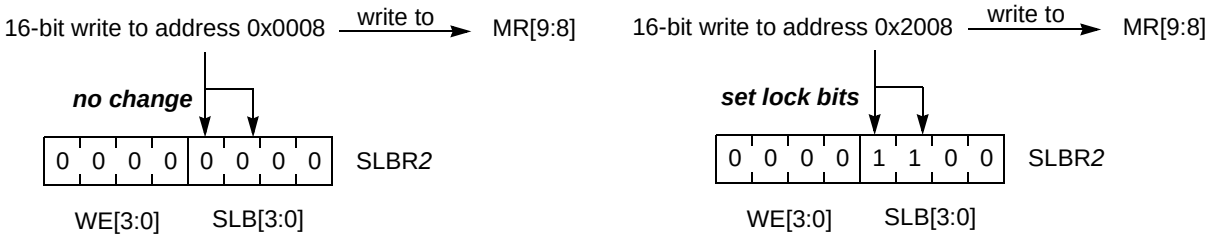


Figure 564. Enable locking via mirror module space (area #3)

When writing to address 0x0008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits remain unchanged (left part of *Figure 561*).

When writing to address 0x2008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits SLBR2.SLB[1:0] are set while the lock bits SLBR2.SLB[3:2] remain unchanged (right part of *Figure 561*).

Figure 565 shows an example where some addresses are protected and some are not.

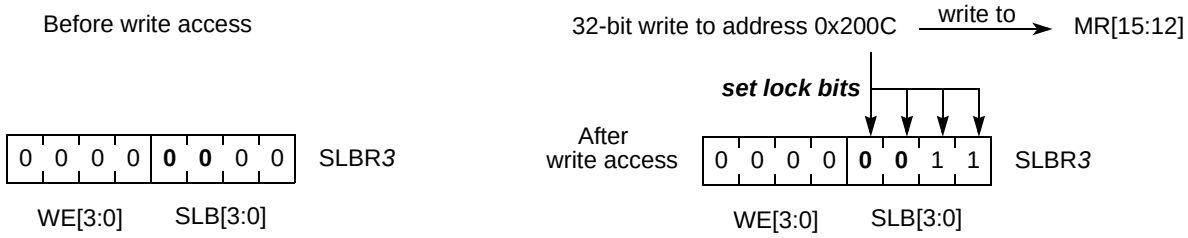


Figure 565. Enable locking for protected and unprotected addresses

In the example in *Figure 565*, addresses 0x0C and 0x0D are unprotected. Therefore their corresponding lock bits SLBR3.SLB[1:0] are always 0b0 (shown in bold). When doing a 32-bit write access to address 0x200C only lock bits SLBR3.SLB[3:2] are set while bits SLBR3.SLB[1:0] stay 0b0.

Note: Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space will not change the lock bits.

Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in *Section : Change lock settings directly via area #4,* and *Section : Enable locking via mirror module space (area #3)* is only possible as long as the GCR[HLB] bit is cleared. Once this bit is set the locking bits can no longer be modified until there was a system reset.

Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to *Figure 557*.

1. If accessing area #1 or area #3, the protection module will pass on any access error from the underlying Module under Protection.
   - If user mode is not allowed, user writes to all areas will assert a transfer error and the writes will be blocked.
   - If accessing the reserved area #2, a transfer error will be asserted.
   - If accessing unimplemented 32-bit registers in area #4 and area #5 a transfer error will be asserted.
   - If writing to a register in area #1 and area #3 with Soft Lock Bit set for any of the affected bytes a transfer error is asserted and the write will be blocked. Also the complete write operation to non-protected bytes in this word is ignored.
   - If writing to a Soft Lock Register in area #4 with the Hard Lock Bit being set a transfer error is asserted.
   - Any write operation in any access mode to area #3 while Hard Lock Bit GCR[HLB] is set

29.2.7 Reset

The reset state of each individual bit is shown in *Section : Registers description.* In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in Supervisor Mode only.

29.3 Software Watchdog Timer (SWT)

29.3.1 Overview

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing sequence. Writing the sequence resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out, a reset is always generated on a second consecutive time-out.

The SWT provides a window functionality. When this functionality is programmed, the servicing action should take place within the defined window. When occurring outside the defined period, the SWT will generate a reset.

29.3.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- The unique SWT counter clock is the undivided low power internal oscillator (IRC 16 MHz), no other clock source can be selected
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- The SWT is started on exit of power-on phase (RGM phase 2) to monitor flash boot sequence phase. It is then reset during RGM phase 3 and optionally enabled when platform reset is released depending on value of flash user option bit 31 (WATCHDOG_EN).

29.3.3 Modes of operation

The SWT supports three device modes of operation: normal, debug and stop. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run. In stop mode, operation of the counter is controlled by the STP bit in the SWT_CR. If the STP bit is set, the counter is stopped in stop mode, otherwise it continues to run. As soon as out of stop mode, SWT will continue from the state it was before entering this mode.

Software watchdog is not available during stand-by. As soon as out of stand-by, the SWT behaves as in a usual “out of reset” situation.

29.3.4 External signal description

The SWT module does not have any external interface signals.

29.3.5 SWT memory map and registers description

The SWT programming model has six 32-bit registers, listed in *Table 538*. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read only registers, incorrect values written to the service register when enabled, accesses to reserved addresses and accesses by masters without permission. If the RIA bit in the SWT_CR is set, the SWT generates a system reset on an invalid access. Otherwise, a bus error is generated. If either the HLK or SLK bits in the SWT_CR are set, then the SWT_CR, SWT_TO and SWT_WN registers are read only.

Table 538. SWT memory map

Offset from SWT_BASE 0xFFF3_8000 (SWT_0) 0x8FF3_8000 (SWT_1)	Register	Location
0x0000	SWT_CR—SWT Control Register	*on page 938*
0x0004	SWT_IR—SWT Interrupt Register	*on page 940*
0x0008	SWT_TO—SWT Time-Out register	*on page 940*
0x000C	SWT_WN—SWT Window Register	*on page 941*
0x0010	SWT_SR—SWT Service Register	*on page 942*
0x0014	SWT_CO—SWT Counter Output register	*on page 942*
0x0018	SWT_SK—SWT Service Key register	*on page 943*
0x001C–0x03FF	Reserved	

SWT Control Register (SWT_CR)

The SWT_CR contains fields for configuring and controlling the SWT. The reset value of this register is device specific. Some devices can be configured to automatically clear the SWT_CR[WEN] bit during the boot process. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Figure 566. SWT Control Register (SWT_CR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP0	MAP1	MAP2	MAP3	MAP4	MAP5	MAP6	MAP7	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1

The default reset value for SWT_0 is 0xFF00_011B, corresponding to MAP1 = 1 (only data bus access allowed), RIA = 1 (reset on invalid SWT access), SLK = 1 (soft lock), CSL = 1 (IRC clock source for counter), FRZ = 1 (freeze on debug), WEN = 1 (watchdog enable). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG_EN) is 0.

For SWT_1 the reset value is 0xFF00_011A and WEN = 0 (watchdog disabled).

Table 539. SWT_CR field descriptions

Field	Description
MAP*n*	Master Access Protection for Master *n*. The platform bus master assignments are device specific. 0 Access for the master is disabled. 1 Access for the master is enabled.
KEY	Keyed Service Mode 0 Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog. 1 Keyed Service Mode, two pseudorandom key values are used to service the watchdog.
RIA	Reset on Invalid Access 0 Invalid access to the SWT generates a bus error. 1 Invalid access to the SWT causes a system reset if WEN = 1.
WND	Window Mode 0 Regular mode, service sequence can be done at any time. 1 Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset 0 Generate a reset on a time-out. 1 Generate an interrupt on an initial time-out, reset on a second consecutive time-out.
HLK	Hard Lock This bit is only cleared at reset. 0 SWT_CR, SWT_TO and SWT_WN are read/write registers if SLK = 0. 1 SWT_CR, SWT_TO and SWT_WN are read only registers.
SLK	Soft Lock This bit is cleared by writing the unlock sequence to the service register. 0 SWT_CR, SWT_TO and SWT_WN are read/write registers if HLK = 0. 1 SWT_CR, SWT_TO and SWT_WN are read only registers.
CSL	Clock Selection Selects the internal 16 MHz IRC oscillator clock that drives the internal timer. CSL bit can be written.The status of the bit has no effect on counter clock selection on the device. 0 System clock. (Not applicable in SPC56xP60/54). 1 Oscillator clock.
STP	Stop Mode Control Allows the watchdog timer to be stopped when the device enters stop mode. 0 SWT counter continues to run in stop mode. 1 SWT counter is stopped in stop mode.
FRZ	Debug Mode Control Allows the watchdog timer to be stopped when the device enters debug mode. 0 SWT counter continues to run in debug mode. 1 SWT counter is stopped in debug mode.
WEN	Watchdog Enabled 0 SWT is disabled. 1 SWT is enabled.

SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Figure 567. SWT Interrupt Register (SWT_IR)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 540. SWT_IR field descriptions

Field	Description
TIF	Time-out Interrupt Flag The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to an initial time-out.

SWT Time-Out register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Figure 568. SWT Time-Out register (SWT_TO)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WTO															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WTO															
W																
Reset	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0

The default counter value (SWT_TO_RST) is 0x0003_A980, which corresponds to approximately 15 ms with the 16 MHz IRC clock.

Table 541. SWT_TO field descriptions

Field	Description
WTO	Watchdog time-out period in clock cycles An internal 32-bit down counter is loaded with this value or 0x0100, whichever is greater when the service sequence is written or when the SWT is enabled.

SWT Window Register (SWT_WN)

The SWT Window (SWT_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Figure 569. SWT Window register (SWT_WN)

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WST															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WST															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 542. SWT_WN field descriptions

Field	Description
WST	Window start value When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service sequence writes used to reset the watchdog timer.

Figure 570. SWT Service Register (SWT_SR)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WSC															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 543. SWT_SR field descriptions

Field	Description
WSC	Watchdog Service Code This field services the watchdog and clears the SWT_CR[SLK] soft lock bit. To service the watchdog, the value 0xA602 followed by 0xB480 is written to the WSC field. To clear the SWT_CR[SLK] soft lock bit, the value 0xC520 followed by 0xD928 is written to the WSC field.

SWT Counter Output register (SWT_CO)

The SWT Counter Output (SWT_CO) register is a read only register that shows the value of the internal down counter when the SWT is disabled.

Figure 571. SWT Counter Output register (SWT_CO)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 544. SWT_CO field descriptions

Field	Description
CNT	Watchdog Count When the watchdog is disabled (SWT_CR.[WEN]=0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for as many as 6 system plus 8 counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Figure 572. SWT Service Register (SWT_SR)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	SK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 545. SWT_SR field descriptions

Field	Description
SK	Service Key.This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SR is (17*SK+3) mod 2^{16}.

29.3.6 Functional description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a control register (SWT_CR), an interrupt register (SWT_IR), a time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR), a counter output register (SWT_CO) and a service key register (SWT_SK).

The SWT_CR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT_CR[WEN] bit. The reset value of the SWT_CR[WEN] bit is device specific. If the reset value of this bit is 1, the watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100 in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_CR[CSL] bit selects which clock (system or

oscillator) is used to drive the down counter. The reset value of the SWT_TO register is device specific.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_CR, SWT_TO, SWT_WN and SWT_SK registers are read only. The hard lock is enabled by setting the SWT_CR[HLK] bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_CR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation which consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the time-out period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT_CR[KEY] bit is zero, the fixed sequence 0xA602, 0xB480 is written to the SWT_SR[WSC] field to service the watchdog. If the SWT_CR[KEY] bit is set, then two pseudorandom keys are written to the SWT_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in *Figure 573*. This algorithm will generate a sequence of 2^{16} different key values before repeating. The state of the key generator is held in the SWT_SK register. For example, if SWT_SK[SK] is 0x0100 then the service sequence keys are 0x1103, 0x2136. In this mode, each time a valid key is written to the SWT_SR register, the SWT_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT_SR[WSC] field, SWT_SK[SK] is 0x2136 and the next key sequence is 0x3499, 0x7E2C.

$$SK_{n+1} = (17*SK_n+3) \bmod 2^{16}$$

Figure 573. Pseudorandom Key Generator

Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require up to three system plus seven counter clock cycles.

If window mode is enabled (SWT_CR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT_CR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window to open due to synchronization logic in the watchdog design. This delay could be up to three system plus four counter clock cycles.

The interrupt then reset bit (SWT_CR[ITR]) controls the action taken when a time-out occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_CR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before

the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a one to the SWT_IR[TIF] bit.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for up to six system plus eight counter clock cycles.

The SWT_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT_CR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

30 Fault Collection and Control Unit (FCCU)

30.1 Introduction

The Fault Collection and Control Unit (FCCU) offers a programmable redundant hardware channel to collect errors and to lead the device in a controlled way to a safe state when a failure is present in the device. No CPU intervention is requested for collection and control operation.

The FCCU offers a systematic approach to manage fault detection and control.

The main functions supported by the module are:

- Configurable and graded fault control via user software control
- Internal reactions:
  - No reset reaction
  - IRQ
- External reaction (failure is reported to the outside world via output pins)
- Watchdog timer for the re-configuration phase
- Configuration lock
- NVM configuration loading
- Self checking capabilities:
  - FCCU internal control logic redundancy
  - Additional parity check for the associated configuration registers

Two classes of faults — critical and non-critical — are identified based on the criticality and the related reactions. Internal (short or long ‘functional’ reset, interrupt request) and external (FCCU_F signaling) reactions are statically defined or programmable based on the fault criticality. The default configuration can be modified only in a specific FCCU state for application/test/debugging purposes.

30.1.1 Glossary and acronyms

Table 546. Acronyms

Term	Description
RCC	Redundancy control checkers
IPS	Internal peripheral system
NMI	Non-maskable interrupts
IRQ	Interrupt request
FSM	Finite state machine
CF	Critical fault
NCF	Non-critical fault
WS	Wait state
SW	Software

Table 546. Acronyms (continued)

Term	Description
NVM	Nonvolatile memory
CPU	Central processing unit

30.2 Main features

- Management of:
  - 32 critical faults
  - 32 non-critical faults
- Hardware or software fault recovery management
- Fault detection collection
- Fault injection (fake faults)
- External reaction via fault-state dedicated system signals (FCCU_F pins)
- Internal (chip) reactions (alarm state): interrupt request
- Internal (chip) reactions (fault state):
- Bi-Stable, Dual-Rail and Time Switching output protocols using FCCU_F pins
- Internal (to the FCCU) watchdog timer for the re-configuration phase
- Configuration lock
- NVM configuration loading
- Self-checking capabilities:
  - FSM redundancy
  - Parity check for the configuration registers

30.3 Block diagram

The top-level diagram of the FCCU is shown in *Figure 574*.

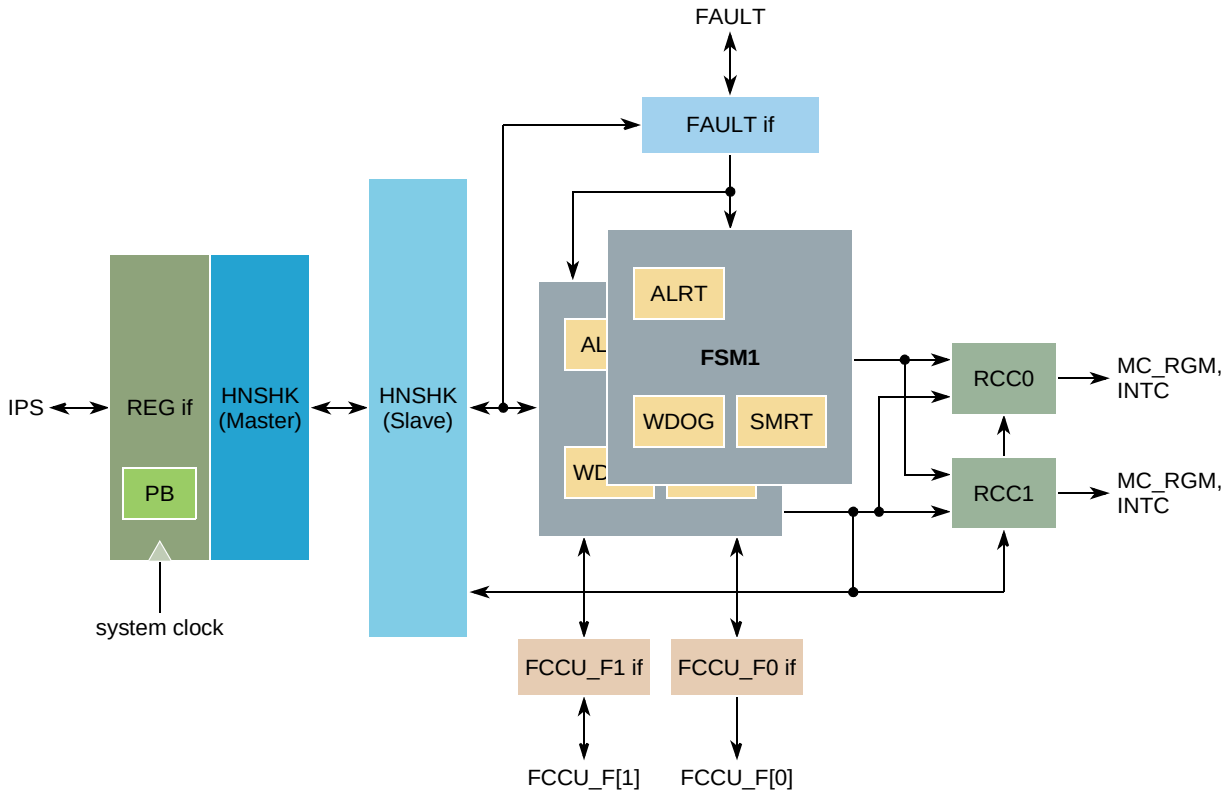


Figure 574. FCCU top-level diagram

As shown in *Figure 574*, the FCCU includes the following sub-modules:

- REG if: it includes the register file, the register interface, the IRQ interface and the parity block (PB) for the configuration registers.
- HNSHK blocks (master and slave blocks): it includes the FSMs to support the handshake between the REG if and the FSM unit due to the usage of 2 asynchronous clocks (system clock and IRCOSC clock).
- Finite State Machine (FSM) units implement the main functions of the FCCU. These units also include also the watchdog timer (WDOG), the ***SAFE*** mode request timer (SMRT), and the alarm timer (ALRT).
- FAULT if: implements the interface for the fault conditioning and management
- FCCU_Fx units implement the output stage to manage the FCCU_F pins.
- RCCx units implement the redundancy control checker to monitor the FSM unit state and its configuration.

30.4 Signal description

The FCCU generates two external signals, FCCU_F[0] and FCCU_F[1]. These are described in *Section 30.7.8, FCCU_F interface*.

30.5 Register interface

The register interface is a slave bus used for configuration purposes via the CPU.

The following bus operations are supported:

- Word (32-bit) data write/read operations to any registers
- Low and high half-words (16-bit, data[0:15] or data[16:31]) data write/read operations to any registers
- Byte (8 bits, data[0:7] or data[8:15] or data[16:23] or data[24:31]) data write/read operations to any registers

Any other operation (free byte enables, misaligned word or half-word access or other operations) are not supported.

The FCCU generates a transfer error in the following cases:

- Any write/read access executed outside the address space of the peripheral
- Any write/read operation different from byte/halfword/word (free byte enables, misaligned access, or other operations) on each register
- Any write access to the configuration registers not executed while the FCCU is not in CONFIG state

The registers of the FCCU are accessible (read/write) in each access mode: user, supervisor, or test.

30.6 Memory map and register description

The FCCU registers are listed in *Table 547*. The contents of the configuration registers (labeled as “W in CONFIG state only” in the Access column) can be locked by the OP16 operation as defined in the FCCU_CTRL register.

Table 547. FCCU memory map

Address offset	Register	Access(1)	Location
0x00	FCCU Control Register (FCCU_CTRL)	R/W always	*on page 951*
0x04	FCCU CTRL Key Register (FCCU_CTRLK)	W always	*on page 954*
0x08	FCCU Configuration Register (FCCU_CFG)	R always; W in CONFIG state only	*on page 955*
0x0C	FCCU CF Configuration Register 0 (FCCU_CF_CFG0)	R always; W in CONFIG state only	*on page 957*
0x10	FCCU CF Configuration Register 1 (FCCU_CF_CFG1)		
0x14	FCCU CF Configuration Register 2 (FCCU_CF_CFG2)		
0x18	FCCU CF Configuration Register 3 (FCCU_CF_CFG3)		
0x1C	FCCU NCF Configuration Register 0 (FCCU_NCF_CFG0)	R always; W in CONFIG state only	*on page 958*
0x20	FCCU NCF Configuration Register 1 (FCCU_NCF_CFG1)		
0x24	FCCU NCF Configuration Register 2 (FCCU_NCF_CFG2)		
0x28	FCCU NCF Configuration Register 3 (FCCU_NCF_CFG3)		

Table 547. FCCU memory map (continued)

Address offset	Register	Access[1]	Location
0x2C	FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0)	R always; W in CONFIG state only	on page 959
0x30	FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1)		
0x34	FCCU CFS Configuration Register 2 (FCCU_CFS_CFG2)		
0x38	FCCU CFS Configuration Register 3 (FCCU_CFS_CFG3)		
0x3C	FCCU CFS Configuration Register 4 (FCCU_CFS_CFG4)		
0x40	FCCU CFS Configuration Register 5 (FCCU_CFS_CFG5)		
0x44	FCCU CFS Configuration Register 6 (FCCU_CFS_CFG6)		
0x48	FCCU CFS Configuration Register 7 (FCCU_CFS_CFG7)		
0x4C	FCCU NCFS Configuration Register 0 (FCCU_NCFS_CFG0)	R always; W in CONFIG state only	on page 961
0x50	FCCU NCFS Configuration Register 1 (FCCU_NCFS_CFG1)		
0x54	FCCU NCFS Configuration Register 2 (FCCU_NCFS_CFG2)		
0x58	FCCU NCFS Configuration Register 3 (FCCU_NCFS_CFG3)		
0x5C	FCCU NCFS Configuration Register 4 (FCCU_NCFS_CFG4)		
0x60	FCCU NCFS Configuration Register 5 (FCCU_NCFS_CFG5)		
0x64	FCCU NCFS Configuration Register 6 (FCCU_NCFS_CFG6)		
0x68	FCCU NCFS Configuration Register 7 (FCCU_NCFS_CFG7)		
0x6C	FCCU CF Status Register 0 (FCCU_CFS0)	R/W always	on page 961
0x70	FCCU CF Status Register 1 (FCCU_CFS1)		
0x74	FCCU CF Status Register 2 (FCCU_CFS2)		
0x78	FCCU CF Status Register 3 (FCCU_CFS3)		
0x7C	FCCU CF Key Register (FCCU_CFK)	W always	on page 963
0x80	FCCU NCF Status Register 0 (FCCU_NCFS0)	R/W always	on page 963
0x84	FCCU NCF Status Register 1 (FCCU_NCFS1)		
0x88	FCCU NCF Status Register 2 (FCCU_NCFS2)		
0x8C	FCCU NCF Status Register 3 (FCCU_NCFS3)		
0x90	FCCU NCF Key Register (FCCU_NCFK)	W always	on page 965
0x94	FCCU NCF Enable Register 0 (FCCU_NCFE0)	R always; W in CONFIG state only	on page 966
0x98	FCCU NCF Enable Register 1 (FCCU_NCFE1)		
0x9C	FCCU NCF Enable Register 2 (FCCU_NCFE2)		
0xA0	FCCU NCF Enable Register 3 (FCCU_NCFE3)		

Table 547. FCCU memory map (continued)

Address offset	Register	Access[1]	Location
0xA4	FCCU NCF Time-out Enable Register 0 (FCCU_NCF_TOE0)	R always; W in CONFIG state only	on page 967
0xA8	FCCU NCF Time-out Enable Register 1 (FCCU_NCF_TOE1)		
0xAC	FCCU NCF Time-out Enable Register 2 (FCCU_NCF_TOE2)		
0xB0	FCCU NCF Time-out Enable Register 3 (FCCU_NCF_TOE3)		
0xB4	FCCU NCF Time-out Register (FCCU_NCF_TO)	R always; W in CONFIG state only	on page 967
0xB8	FCCU CFG Timeout Register (FCCU_CFG_TO)	R always; W always except CONFIG state	on page 968
0xC0	FCCU Status Register (FCCU_STAT)	R always	on page 969
0xC4 - 0xC7	Reserved		
0xD8	FCCU CF Fake Register (FCCU_CFF)	W always	on page 971
0xDC	FCCU NCF Fake Register (FCCU_NCFF)	W always	on page 972
0xE0	FCCU IRQ Status Register (FCCU_IRQ_STAT)	R/W always	on page 972
0xE4	FCCU IRQ Enable Register (FCCU_IRQ_EN)	R/W always	on page 973
0xE8	FCCU XTMR Register (FCCU_XTMR)	R always	on page 974
0xEC	FCCU MCS Register (FCCU_MCS)	R always	on page 975

1. In this column, R/W = read/write, R = read-only, and W = write-only.

30.6.1 FCCU Control Register (FCCU_CTRL)

The FCCU_CTRL register allows execution of the following operations:

- Move the FCCU state from the NORMAL state into the CONFIG state
- Move the FCCU state from the CONFIG state into the NORMAL state
- Read or clear the CF status register
- Read or clear the NCF status register
- Read the FCCU FSM status register
- Read or clear the FCCU freeze registers
- Lock the FCCU configuration registers
- Read the ALARM timer
- Read the SMRT timer
- Read the Watchdog timer
- Load the NVM configuration (only for test purposes)

Some critical operations (OP1, OP2, OP16, and OP31) require a key as defined in the FCCU_CTRLK register.

Figure 575. FCCU Control Register (FCCU_CTRL)

Offset: 0x00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	NVML	OPS		0	OPR				
W																
Reset	0	0	0	0	0	0	0	0/1(1)	0/1(1)	0/1(1)	0	0	0	0	0	0

1. 1 when NVM interface is used, 0 otherwise.

Table 548. FCCU_CTRL field descriptions

Field	Description
NVML	NVM configuration loaded 0 No NVM configuration loaded. 1 NVM configuration loaded. At the end of the reset PHASE3, NVML = 1 and OPS = 11 if the NVM interface of the FCCU is correctly driven by the SSCM interface. This bit can be read and cleared (via OP15 operation) by the software.
OPS	Operation status 00 Idle. 01 In progress. 10 Aborted. 11 Successful. This bit can be read and cleared (via OP15 operation) by the software.

Table 548. FCCU_CTRL field descriptions (continued)

Field	Description
OPR	Operation run 00000 No operation [OP0]. 00001 Set the FCCU into the CONFIG state [OP1]. 00010 Set the FCCU into the NORMAL state [OP2]. 00011 Read the FCCU state (refer to the FCCU_STAT register) [OP3]. 00100 Read the FCCU frozen status flags [OP4]. 00101 Read the FCCU frozen status flags [OP5]. 00110 Read the FCCU frozen status flags [OP6]. 00111 Read the FCCU frozen status flags [OP7]. 01000 Read the FCCU frozen status flags [OP8]. 01001 Read the CF status register (refer to the FCCU_CFS register) [OP9]. 01010 Read the NCF status register (refer to the FCCU_NCFS register) [OP10]. 01011 CF status clear operation in progress (refer to the FCCU_CFS register) [OP11]. 01100 NCF status clear operation in progress (refer to the FCCU_NCFS register) [OP12]. 01101 Clear the freeze status registers (refer to the freeze registers) [OP13]. 01110 CONFIG to NORMAL FCCU state (configuration timeout) in progress [OP14]. 01111 Clear the operation status (OPS = Idle, NVML = 0) [OP15]. 10000 Lock the FCCU configuration [OP16]. 10001 Read the ALARM timer (refer to the FCCU_XTMR register) [OP17]. 10010 Read the SMRT timer (refer to the FCCU_XTMR register) [OP18]. 10011 Read the CFG timer (refer to the FCCU_XTMR register) [OP19]. 10100–11111 Reserved [OP20–OP30]. 11111 Run the NVM loading operation (only for test purposes) [OP31]. The software must not modify the OPR field until the completion of the operation (any write operation will be ignored until then). After the operation has been completed, the OPS field is set and the OPR field is automatically cleared (OPR = 000). Your software must not program the following opcodes: – OP11 and OP12 (these opcodes are automatically selected when the FCCU_CFS or FCCU_NCFS registers are cleared by a write-clear operation into the related register) – OP14 (This opcode is automatically selected when the timeout occurs [FCCU_CFG_TO] during the configuration procedure. In this case, the FCCU state is automatically forced in NORMAL mode setting the default configuration. In this phase any write operation to the FCCU configuration registers is inhibited.) – OP20–OP30 (these are reserved; if you attempt to use them, they will return an ABORT response without any side effect) The ABORT response occurs in the following cases: – wrong access (missing or wrong key) to the FCCU_CFS register (clear operation OP11) – wrong access (missing or wrong key) to the FCCU_NCFS register (clear operation OP12) – wrong access (missing or wrong key) to the FCCU_CTRL register (OP1, OP2, OP16 operation) – OP1 (CONFIG command) execution when FCCU state ≠ NORMAL or configuration locked – OP20–OP30 (RESERVED operations) execution The OP31 opcode executes the NVM configuration loading via the software. It should be used only for test/debug purposes.

30.6.2 FCCU CTRL Key Register (FCCU_CTRLK)

The FCCU_CTRLK register implements the key access for the operations OP1, OP2, OP16, OP31 according to the following sequence:

1. Write the key into the FCCU_CTRLK register.
2. Write the FCCU_CTRL register (operations OP1, OP2, OP16, or OP31).

The FCCU_CTRLK register is not readable. A read operation always returns 0x0000_0000. The key must be written by a word (32-bit) data write operation.

Figure 576. FCCU CTRL Key Register (FCCU_CTRLK)

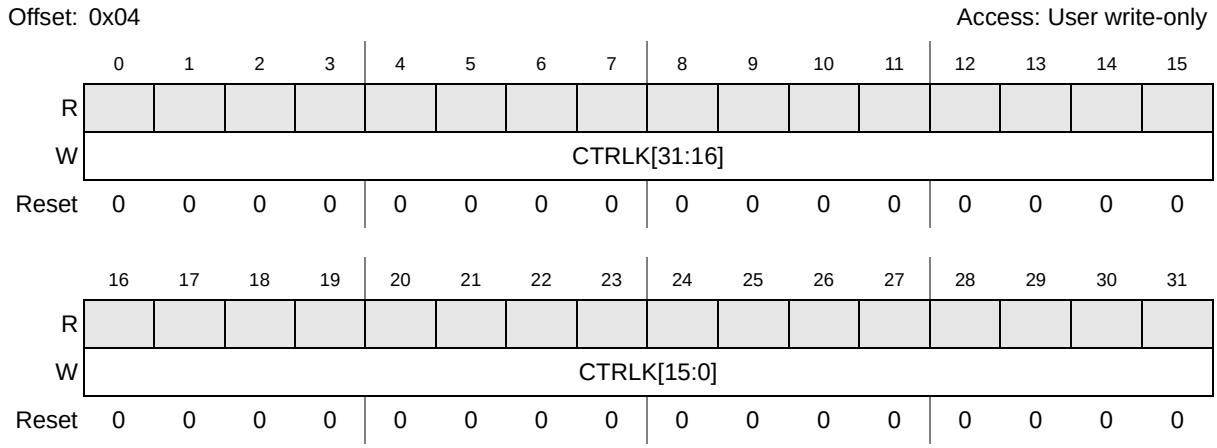


Table 549. FCCU_CTRLK field descriptions

Field	Description
CTRLK	Control register key: **CTRLK value** / **Function** 0x9137_56AF – Key for operation OP1 0x825A_132B – Key for operation OP2 0x7ACB_32F0 – Key for operation OP16 0x29AF_8752 – Key for operation OP31

30.6.3 FCCU Configuration Register (FCCU_CFG)

The FCCU_CFG register is accessible in write mode only in the CONFIG state. It defines the global configuration for the FCCU.

Figure 577. FCCU Configuration Register (FCCU_CFG)

Offset: 0x008

Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	RCCE1	RCCE0	SMRT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	FCCU_CFG.CM	FCCU_CFG.SM	FCCU_CFG.PS	FCCU_CFG.FOM			FCCU_CFG.FOP					
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

1. Writable only in the CONFIG state

Table 550. FCCU_CFG field descriptions

Field	Description
RCCE1	RCC1 enable 0 RCC1 disabled. 1 RCC1 enabled. *Note: In case a single checker (RCC1 or RCC0 unit) is enabled, both the checkers assert a fault condition (destructive reset).*
RCCE0	RCC0 enable 0 RCC0 disabled. 1 RCC0 enabled. *Note: In case a single checker (RCC1 or RCC0 unit) is enabled, both the checkers assert a fault condition (destructive reset).*
SMRT	***SAFE** Mode Request Timer* 0000 ***SAFE*** mode request delay = 1 period (IRCOSC clock). 0001 ***SAFE*** mode request delay = 2 periods (IRCOSC clock). 0010 ***SAFE*** mode request delay = 4 periods (IRCOSC clock). ... 1111 ***SAFE*** mode request delay = 32768 periods (IRCOSC clock).
FCCU_CFG.CM	Configuration mode 0 Configuration labelling: a specific FCCU_F configuration is assigned in CONFIG state. 1 Configuration transparency: the FCCU_F protocol is the same in CONFIG and NORMAL states.

Table 550. FCCU_CFG field descriptions (continued)

Field	Description
FCCU_CFG.SM	Switching mode 0 FCCU_F protocol (dual-rail, time-switching) slow switching mode. 1 FCCU_F protocol (dual-rail, time-switching) fast switching mode. SM has no effect on the bi-stable protocol.
FCCU_CFG.PS	Polarity selection 0 FCCU_F[1] active high, FCCU_F[0] active high. 1 FCCU_F[1] active low, FCCU_F[0] active low.
FCCU_CFG.FOM	Fault Output Mode selection 000 Dual-Rail (default state; FCCU_F[1:0]= outputs). 001 Time Switching (FCCU_F[1:0]= outputs). 010 Bi-Stable (FCCU_F[1:0]= outputs). 011 Reserved. 100 Reserved. 101 Test0 (FCCU_F[0] = input, FCCU_F[1]= output) 110 Test1 (FCCU_F[0] = output, FCCU_F[1]= output) 111 Test2 (FCCU_F[0] = output, FCCU_F[1]= input) *Note: In Testn mode, a simple double-stage resynchronization stage is used to resynchronize the FCCU_F input/outputs on the system/IRCOSC clock.*
FCCU_CFG.FOP	Fault Output Prescaler FOP defines the prescaler setting used to generate the FCCU_F protocol frequency. 00 0000 Input clock frequency (IRCOSC clock) is divided by 2 00 0001 Input clock frequency (IRCOSC clock) is divided by 4 00 0010 Input clock frequency (IRCOSC clock) is divided by 6 00 0011 Input clock frequency (IRCOSC clock) is divided by 8 00 0100 Input clock frequency (IRCOSC clock) is divided by 10 00 0101 Input clock frequency (IRCOSC clock) is divided by 12 00 0110 Input clock frequency (IRCOSC clock) is divided by 14 The following equation gives the FCCU_F frequency: $F_{FCCU\_F} = F_{IRCOSC} / (1024 \times (FOP + 1) \times 2)$

30.6.4 FCCU CF Configuration Register (FCCU_CF_CFG0..3)

The FCCU_CF_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each critical fault in terms of fault recovery management.

The configuration depends on the type of signaling following a fault event. Hardware recoverable faults should be configured only if a previous latching stage captures and hold the physical fault otherwise the fault can be lost. All the other faults should be configured as SW fault.

Figure 578. FCCU CF Configuration Register (FCCU_CF_CFG0..3)

Offset: 0x00C–0x018 (4 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFC 31	CFC 30	CFC 29	CFC 28	CFC 27	CFC 26	CFC 25	CFC 24	CFC 23	CFC 22	CFC 21	CFC 20	CFC 19	CFC 18	CFC 17	CFC 16
W																
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFC 15	CFC 14	CFC 13	CFC 12	CFC 11	CFC 10	CFC 9	CFC 8	CFC 7	CFC 6	CFC 5	CFC 4	CFC 3	CFC 2	CFC 1	CFC 0
W																
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_CF_CFG0; 0 otherwise

Table 551. FCCU_CF_CFG0..3 field descriptions

Field	Description
CFCx	Critical fault configuration 0 Hardware recoverable fault. 1 Software recoverable fault. The critical fault configuration defines the fault recovery mode. Hardware-recoverable faults are self recovered (status flag clearing) if the root cause has been removed. SW recoverable faults are recovered (status flag clearing) by software clearing the related status flag.

30.6.5 FCCU NCF Configuration Register (FCCU_NCF_CFG0..3)

The FCCU_NCF_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each non-critical fault in terms of fault recovery management. The configuration depends on the type of signaling of a fault event. HW recoverable faults should be configured only if a previous latching stage captures and hold the physical fault otherwise the fault can be lost. All the other faults should be configured as SW fault.

Figure 579. FCCU NCF Configuration Register (FCCU_NCF_CFG0..3)

Offset: 0x01C–0x028 (4 registers)

Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFC31	NCFC30	NCFC29	NCFC28	NCFC27	NCFC26	NCFC25	NCFC24	NCFC23	NCFC22	NCFC21	NCFC20	NCFC19	NCFC18	NCFC17	NCFC16
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFC15	NCFC14	NCFC13	NCFC12	NCFC11	NCFC10	NCFC9	NCFC8	NCFC7	NCFC6	NCFC5	NCFC4	NCFC3	NCFC2	NCFC1	NCFC0
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_NCF_CFG0; 0 otherwise

Table 552. FCCU_NCF_CFG0..3 field descriptions

Field	Description
NCFCx	*Non-critical fault configuration* 0: HW recoverable fault 1: SW recoverable fault The non-critical fault configuration defines the fault recovery mode. HW recoverable faults are self recovered (status flag clearing and related) if the root cause has been removed. SW recoverable faults are recovered (status flag clearing) by SW clearing of the related status flag. This register can be read and written by the software.

30.6.6 FCCU CFS Configuration Register (FCCU_CFS_CFG0..7)

The FCCU_CF_FS_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each critical fault in terms of fault reaction (short or long ‘functional’ reset) when it is the root cause for the FAULT state transition.

Figure 580. FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0)

Offset: 0x02C — Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC15		CFSC14		CFSC13		CFSC12		CFSC11		CFSC10		CFSC9		CFSC8	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC7		CFSC6		CFSC5		CFSC4		CFSC3		CFSC2		CFSC1		CFSC0	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

1. Writable only in the CONFIG state

Figure 581. FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1)

Offset: 0x030 — Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC15		CFSC14		CFSC13		CFSC12		CFSC11		CFSC10		CFSC9		CFSC8	
Reset	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC7		CFSC6		CFSC5		CFSC4		CFSC3		CFSC2		CFSC1		CFSC0	
Reset	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

1. Writable only in the CONFIG state

Figure 582. FCCU CFS Configuration Register 2..7 (FCCU_CFS_CFG2..7)

Offset: 0x034–0x048 (6 registers)

Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC15		CFSC14		CFSC13		CFSC12		CFSC11		CFSC10		CFSC9		CFSC8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC7		CFSC6		CFSC5		CFSC4		CFSC3		CFSC2		CFSC1		CFSC0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Writable only in the CONFIG state

Table 553. FCCU_CFS_CFG0..7 field descriptions

Field	Description
CFSCx	*Critical fault state configuration* 00: No reset reaction 01: Short functional reset (FAULT state reaction) 10: Long functional reset (FAULT state reaction) 11: No reset reaction This register can be read and written by the software.

30.6.7 FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7)

The FCCU_NCF_FS_CFGx register is accessible in write mode only in the CONFIG state. It contains the configuration of each non-critical fault in terms of fault reaction (short or long ‘functional’ reset) when it is the root cause for the FAULT state transition.

Figure 583. FCCU NCFS Configuration Register (FCCU_NCFS_CFG0..7)

Offset: 0x04C–0x068 (8 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFSC15		NCFSC14		NCFSC13		NCFSC12		NCFSC11		NCFSC10		NCFSC9		NCFSC8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFSC7		NCFSC6		NCFSC5		NCFSC4		NCFSC3		NCFSC2		NCFSC1		NCFSC0	
Reset	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0	0/1(2)	0

1. Writable only in the CONFIG state
2. 1 for FCCU_NCFS_CFG0; 0 otherwise

Table 554. FCCU_NCFS_CFG0..7 field descriptions

Field	Description
NCFSCx	*Non-critical fault state configuration* 00: No reset reaction 01: Short ‘functional’ reset (FAULT state reaction) 10: Long ‘functional’ reset (FAULT state reaction) 11: No reset reaction This register can be read and written by the software.

30.6.8 FCCU CF Status Register (FCCU_CFS0..3)

The FCCU_CFSx register contains the latched fault indication collected from the critical fault sources. Faults are latched even if the FCCU is in the CONFIG state and independently from the enabling or reactions programmed for the CF.
No reactions are executed until the FCCU moves in the NORMAL state. FCCU reacts and moves from the NORMAL or ALARM state into the FAULT state if a critical fault is triggered.
The status bits of the FCCU_CFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- to write the proper key into the FCCU_CFK register
- to clear the status (flag) bit CFSx → the opcode OP11 is automatically set into the FCCU_CTRL.OPR field

- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_CFSx register in order to verify the effective deletion and in case of failure to repeat the sequence

As result of the above sequence, in addition the FAULT interface provides support to clear the external FAULT root.

The FCCU moves from the FAULT state into the NORMAL state if the source fault which caused the transition into the FAULT state has been removed (HW recoverable fault) or cleared via SW (SW recoverable fault). Concurrently the FAULT interface provides support to clear the FAULT root. In case of nested faults that are not all recovered, the FCCU remains into the FAULT state or moves into the ALARM state.

The SW application executes the FCCU_CFSx read operation by the following sequence:

- to set the OP9 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_CFSx register

The following errors are ignored:

- to write a wrong key into the FCCU_CFK register
- to attempt to clear a HW recoverable error

Figure 584. FCCU CF Status Register (FCCU_CFS0..3)

Offset: 0x06C–0x078 (4 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS31	CFS30	CFS29	CFS28	CFS27	CFS26	CFS25	CFS24	CFS23	CFS22	CFS21	CFS20	CFS19	CFS18	CFS17	CFS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFS15	CFS14	CFS13	CFS12	CFS11	CFS10	CFS9	CFS8	CFS7	CFS6	CFS5	CFS4	CFS3	CFS2	CFS1	CFS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Register FCCU_CFS0 shows status of CF[0:31].

Register FCCU_CFS1 shows status of CF[37] (bit26). All other register bits of FCCU_CFS1 are unused.

Register FCCU_CFS2 and Register FCCU_CFS3 are implemented but are not used. They do not show any status.

Table 555. FCCU_CFS0..3 field descriptions

Field	Description
CFSx	*Critical fault status* 0: No critical fault latched 1: Critical fault latched The status bits related to the critical fault configured as HW recoverable faults are read-only and the flag is self cleared when the fault source is removed. The status bits related to the critical fault configured as SW recoverable faults are write-clear and the SW application can recover from a faulty condition.

30.6.9 FCCU CF Key Register (FCCU_CFK)

The FCCU_CFK register implements the key access to clear the status flags of the FCCU_CFSx register.

The status bits of the FCCU_CFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- Write the CFCK key into the FCCU_CFK register
- Clear the status (flag) bit CFSx

Note: The key must be written for each FCCU_CFSx clear operation.

The FCCU_CFK register is not readable, a 0x00000000 value is always returned in case of read operation.

Figure 585. FCCU CF Key Register (FCCU_CFK)

Offset: 0x07C Access: User write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	CFK[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	CFK[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 556. FCCU_CFK field descriptions

Field	Description
CFK	*Critical fault key* = 0x618B7A50

30.6.10 FCCU NCF Status Register (FCCU_NCFS0..3)

The FCCU_NCFSx register contains the latched fault indication collected from the non-critical fault sources. Faults are latched also in the CONFIG state and independently from

the enabling or reactions programmed for the NCF.
No reactions are executed until the FCCU moves in the NORMAL state.
FCCU reacts and moves from the NORMAL state into the ALARM state only if the respective enable bit for a fault is set in the FCCU_NCFEx register and the respective enable bit for the time-out is set in the FCCU_TOEx register.
FCCU reacts and moves from the NORMAL or ALARM state into the FAULT state if the respective enable bit for a fault is set in the FCCU_NCFEx register and the respective enable bit for the time-out is disabled in the FCCU_TOEx register.
FCCU reacts and moves from the ALARM state into the FAULT state if the time-out (FCCU_TO register) is elapsed before to recovery the fault.

The time-out is stopped only when the FCCU returns in the NORMAL state.

The status bits of the FCCU_NCFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- Write the proper key into the FCCU_NCFK register
- Clear the status (flag) bit NCFSx → the opcode OP12 is automatically set into the FCCU_CTRL.OPR field
- Wait for the completion of the operation (FCCU_CTRL.OPS field)
- Read the FCCU_NCFSx register in order to verify the effective deletion and in case of failure to repeat the sequence

As result of the above sequence, in addition the FAULT interface provides support to clear the external FAULT root.

The FCCU moves from the FAULT or ALARM state into the NORMAL state if all the source faults which caused the transition into the FAULT state has been removed (HW recoverable fault) or cleared via SW (SW recoverable fault). In case of nested faults that are not all recovered, the FCCU will remain in the FAULT or ALARM state.

The SW application executes the FCCU_NCFSx read operation by the following sequence:

- to set the OP10 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_NCFSx register

In case of re-configuration of the FCCU (CONFIG state), before to return in NORMAL state the pending status bits into the FCCU_NCFSx must be cleared in order to avoid a false transition in ALARM/FAULT state.

The following errors are ignored:

- to write a wrong key into the FCCU_NCFK register
- to attempt to clear a HW recoverable error

Figure 586. FCCU NCF Status Register (FCCU_NCFS0..3)

Offset: 0x080–0x08C (4 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCFS31	NCFS30	NCFS29	NCFS28	NCFS27	NCFS26	NCFS25	NCFS24	NCFS23	NCFS22	NCFS21	NCFS20	NCFS19	NCFS18	NCFS17	NCFS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCFS15	NCFS14	NCFS13	NCFS12	NCFS11	NCFS10	NCFS9	NCFS8	NCFS7	NCFS6	NCFS5	NCFS4	NCFS3	NCFS2	NCFS1	NCFS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 557. FCCU_NCFS0..3 field descriptions

Field	Description
NCFSx	*Non-critical fault status* 0: No “non-critical” fault latched 1: “non-critical fault” latched The status bits related to the non-critical fault configured as HW recoverable faults are read-only and the flag is self cleared when the fault source is removed. The status bits related to the critical fault configured as SW recoverable faults are write-clear and the SW application can recover from a faulty condition.

30.6.11 FCCU NCF Key Register (FCCU_NCFK)

The FCCU_NCFK register implements the key access to clear the status flags of the FCCU_NCFSx register.
The status bits of the FCCU_NCFSx register, configured as SW recoverable faults, can be cleared by the following locked sequence:

- to write the key into the FCCU_NCFK register
- to clear the status (flag) bit NCFSx

Note: The key must be written for reach FCCU_NCFSx clear operation.

The FCCU_NCFK register is not readable, a 0x00000000 value is always returned in case of read operation.

Figure 587. FCCU NCF Key Register (FCCU_NCFK)

Offset: 0x090 Access: User write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	NCFK[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	NCFK[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 558. FCCU_NCFK field descriptions

Field	Description
NCFK	*Critical fault key* = 0xAB3498FE

30.6.12 FCCU NCF Enable Register (FCCU_NCFE0..3)

The FCCU_NCFEx register enables the critical fault sources to allow a transition from the NORMAL into the FAULT or ALARM state. In case of fault masking, the respective status bit into the FCCU_NCFSx register is anyway set (for debugging purposes), only the reaction is masked.

The FCCU_NCFEs register is accessible in write mode only in the CONFIG state.

Figure 588. FCCU NCF Enable Register (FCCU_NCFE0..3)

Offset: 0x094–0x0A0 (4 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFE31	NCFE30	NCFE29	NCFE28	NCFE27	NCFE26	NCFE25	NCFE24	NCFE23	NCFE22	NCFE21	NCFE20	NCFE19	NCFE18	NCFE17	NCFE16
Reset	0	0	0	0	0	0	0	0/1(2)	0/1(2)	0/1(2)	0	0/1(2)	0	0	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFE15	NCFE14	NCFE13	NCFE12	NCFE11	NCFE10	NCFE9	NCFE8	NCFE7	NCFE6	NCFE5	NCFE4	NCFE3	NCFE2	NCFE1	NCFE0
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0	0	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_NCFE0; 0 otherwise

Table 559. FCCU_NCFE0..3 field descriptions

Field	Description
NCFEx	*Non-critical fault enable* 0: No actions following the respective critical fault assertion 1: FCCU moves to ALARM or FAULT state

30.6.13 FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3)

The FCCU_NCFTOEx register enables a transition from the NORMAL state into the ALARM state if the respective non-critical fault is enabled (NCFEx and NCFTOEx are set). In case the respective time-out is disabled (NCFTOEx is cleared) and the non-critical fault is enabled (NCFEx is set) the FCCU moves into the FAULT state if the related non-critical fault is asserted. The timer (preset with the time-out value defined by FCCU_TO register) is started when the FCCU moves into the ALARM state. If the fault is not recovered within the time-out the FCCU moves from the ALARM state to the FAULT state.

The FCCU_NCFTOEx register is accessible in write mode, only in the CONFIG state.

Figure 589. FCCU NCF Time-out Enable Register (FCCU_NCF_TOE0..3)

Offset: 0x0A4–0x0B0 (4 registers) Access: User read/write(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFTOE31	NCFTOE30	NCFTOE29	NCFTOE28	NCFTOE27	NCFTOE26	NCFTOE25	NCFTOE24	NCFTOE23	NCFTOE22	NCFTOE21	NCFTOE20	NCFTOE19	NCFTOE18	NCFTOE17	NCFTOE16
Reset	0	0	0	0	0	0	0	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0	0/1(2)	0/1(2)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFTOE15	NCFTOE14	NCFTOE13	NCFTOE12	NCFTOE11	NCFTOE10	NCFTOE9	NCFTOE8	NCFTOE7	NCFTOE6	NCFTOE5	NCFTOE4	NCFTOE3	NCFTOE2	NCFTOE1	NCFTOE0
Reset	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)	0/1(2)

1. Writable only in the CONFIG state
2. 1 for FCCU_NCF_TOE0; 0 otherwise

Table 560. FCCU_NCF_TOE0..3 field descriptions

Field	Description
NCFTOEx	*Non-critical fault time-out enable* 0: FCCU moves into the FAULT state if the respective fault is disabled (NCFEx is cleared) 1: FCCU moves into the ALARM state if the respective fault is enabled (NCFEx is set)

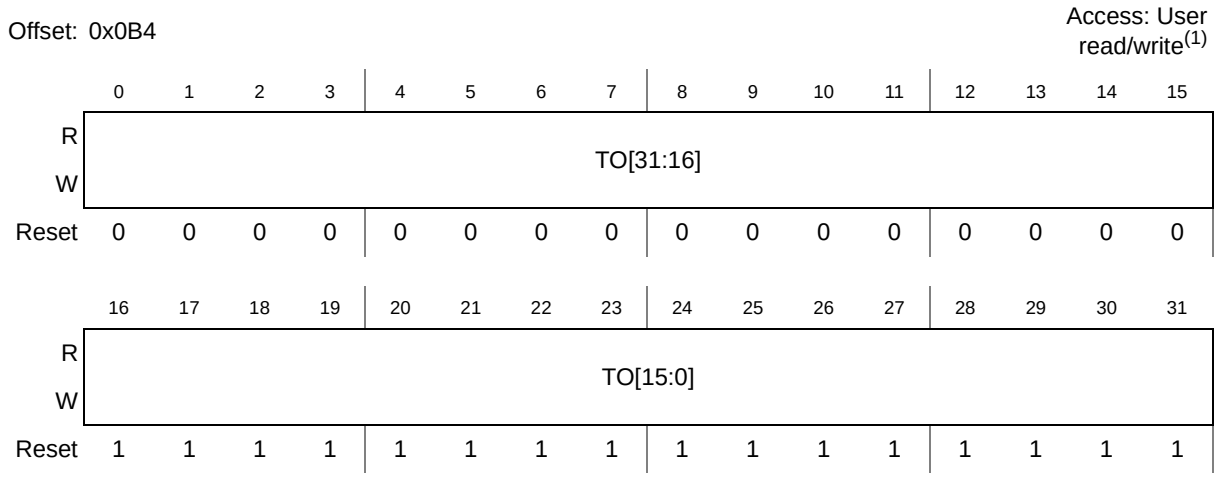
30.6.14 FCCU NCF Time-out Register (FCCU_NCF_TO)

The FCCU_NCF_TO register defines the preset value of the timer for the recovery of the non-critical faults (enabled). Once FCCU enters in ALARM state, following the assertion of a

non-critical fault enabled (NCFEx and NCFTOEx are set), the timer starts the count down. If the fault is not recovered within the time-out the FCCU moves from the ALARM state to the FAULT state.

The FCCU_NCF_TO register is accessible in write mode, only in the CONFIG state.

Figure 590. FCCU NCF Time-out Register (FCCU_NCF_TO)



1. Writable only in the CONFIG state

Table 561. FCCU_NCF_TO field descriptions

Field	Description
TO	*Non-critical fault timeout* $Timeout = (TO) \times T_{RC16MHz}$

30.6.15 FCCU CFG Timeout Register (FCCU_CFG_TO)

The FCCU_CFG_TO register defines the preset value of the watchdog timer for the recovery from the CONFIG state. Once FCCU enters in CONFIG state, following a SW request (OP1 opcode), the watchdog timer is initialized and starts the countdown if the reset is not asserted.
If the configuration is not completed within the time-out, the FCCU moves automatically from the CONFIG state to the NORMAL state and the default values for the configuration register is restored. The watchdog time-out is clocked with the IRCOSC clock (16 MHz). The default time-out value is 4,096 ms.

The FCCU_CFG_TO register is accessible in write mode, in any state excluded the CONFIG state as follows the execution of the OP1 opcode (NORMAL to CONFIG state) and until the completion of the OP2 opcode (CONFIG to NORMAL state).
In case of watchdog time-out the FCCU_CFG_TO register is not accessible until the OP14 operation (CONFIG to NORMAL) has been completed.

Figure 591. FCCU CFG Timeout Register (FCCU_CFG_TO)

Offset: 0x0B8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	TO		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Table 562. FCCU_CFG_TO field descriptions

Field	Description
TO	*Configuration time-out* $Timeout = T_{RC16MHz} \times 2^{(TO+10)}$ 000: Time-out = 64 µs .... 111: Time-out = 8,192 ms

30.6.16 FCCU Status Register (FCCU_STAT)

The FCCU_STAT register includes the FCCU status for debugging/test purposes. The FCCU finite state machine operates by the IRCOSC clock asynchronous with the system clock. The FCCU status read operation requires a safe mechanism operated by a HW/SW synchronization sequence. The SW application executes a FCCU status read operation by the following sequence:

- to set the OP3 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU status (FCCU_STAT register)

Figure 592. FCCU Status Register (FCCU_STAT)

Offset: 0x0C0 Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	STATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 563. FCCU_STAT field descriptions

Field	Description
STATUS	*FCCU Status* 000: NORMAL state. 001: CONFIG state 010: ALARM state 011: FAULT state Other: UNKNOWN state This bit can be read by the software.

30.6.17 FCCU CF Fake Register (FCCU_CFF)

The FCCU_CFF register contains a unique code to set a “critical fault” in mutually exclusive mode by the external FAULT interface. It allows the SW emulation of the critical faults, by the injection of the fault directly in the FAULT root, in order to verify the entire path and reaction. The fault injection mechanism is optional. The reaction following a fake critical fault cannot be masked. The FCCU_CFF is a write-only register with a set of codes corresponding to each critical fault injection.

Figure 593. FCCU CF Fake Register (FCCU_CFF)

Offset: 0x0D8 Access: User write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0							
W										FCFC						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 564. FCCU_CFF field descriptions

Field	Description
FCFC	*Fake critical fault code* 00h: Fake critical fault injection at critical fault source 0 01h: Fake critical fault injection at critical fault source 1 02h: Fake critical fault injection at critical fault source 2 .. 1Fh: Fake critical fault injection at critical fault source 31 others: No fault injection These bits are always read as ‘0’ by software.

30.6.18 FCCU NCF Fake Register (FCCU_NCFF)

The FCCU_NCFF register contains a unique code to set a “non-critical fault” in mutually exclusive mode by the external FAULT interface. It allows the SW emulation of the non-critical faults, by the injection of the fault directly in the FAULT root, in order to verify the entire path and reaction. The fault injection mechanism is optional. The reaction following a fake non-critical fault can be masked. The FCCU_NCFF is a write-only register with a set of codes corresponding to each non-critical fault injection.

Figure 594. FCCU NCF Fake Register (FCCU_NCFF)

Offset: 0x0DC Access: User write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0							
W										FNCFC						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 565. FCCU_NCFF field descriptions

Field	Description
FNCFC	*Fake non-critical fault code* 00h: Fake non-critical fault injection at non-critical fault source 0 01h: Fake non-critical fault injection at non-critical fault source 1 02h: Fake non-critical fault injection at non-critical fault source 2 .. 1Fh: Fake non-critical fault injection at non-critical fault source 31 others: No fault injection These bits are always read as ‘0’ by software.

30.6.19 FCCU IRQ Status Register (FCCU_IRQ_STAT)

The FCCU_IRQ_STAT register defines the FCCU interrupt status register related to the following events:

- Configuration time-out error
- Alarm interrupt

The external interrupt is asserted if any interrupt status bit of the FCCU_IRQ_STAT is set and the respective enable bit of the FCCU_IRQ_EN register is also set.

The ALARM interrupts is asserted and cleared according to the FCCU state. The status bits of the FCCU_IRQ_STAT trace the status of the related interrupt lines.

Figure 595. FCCU IRQ Status Register (FCCU_IRQ_STAT)

Offset: 0x0E0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ALRM_STAT	CFG_TO_STAT
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 566. FCCU_IRQ_STAT field descriptions

Field	Description
CFG_TO_STAT	*Configuration Time-out Status* 0: No configuration time-out error 1: Configuration time-out error This bit can be read and cleared by the software.
ALRM_STAT	*Alarm Interrupt Status* 0: Alarm interrupt is OFF 1: Alarm interrupt is ON This bit can be only read by the software.

30.6.20 FCCU IRQ Enable Register (FCCU_IRQ_EN)

The FCCU_IRQ_EN register defines the FCCU interrupt enable register related to the following events:

- Configuration time-out error

The external interrupt is asserted if any interrupt status bit of the FCCU_IRQ_STAT is set and the respective enable bit of the FCCU_IRQ_EN register is also set.

Figure 596. FCCU IRQ Enable Register (FCCU_IRQ_EN)

Offset: 0x0E4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CFG_TO_IEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 567. FCCU_IRQ_EN field descriptions

Field	Description
CFG_TO_IEN	*Configuration Time-out Interrupt Enable* 0: Configuration time-out interrupt disabled 1: Configuration time-out interrupt enabled This bit can be read and written by the software.

30.6.21 FCCU XTMR Register (FCCU_XTMR)

The FCCU_XTMR register contains the read values of the Alarm, Watchdog or Safe Mode Request Timer. These timers are clocked on the IRCOSC clock.

The SW application executes the timer read operation by the following sequence:

- to set the OP17 or OP18 or OP19 operation into the FCCU_CTRL.OPR field
- to wait for the completion of the operation (FCCU_CTRL.OPS field)
- to read the FCCU_XTMR register

Table 568. Timer state/value

TIMER	CONFIG state	NORMAL state	ALARM state	FAULT state
ALARM	00000000h	Initial value	Running	Idle/End of count
SMRT	00000001h	Initial value	—	Running/End of count
CFG	Running	0001FFFFh	0001FFFFh	0001FFFFh

Figure 597. FCCU XTMR Register (FCCU_XTMR)

Offset: 0x0E8 Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	XTMR[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	XTMR[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 569. FCCU_XTMR field descriptions

Field	Description
XTMR	*Alarm/Watchdog/Safe request timer* The current timer value is measured in IRCOSC clock cycles. These bits can be read by the software.

30.6.22 FCCU MCS Register (FCCU_MCS)

The FCCU_MCS register contains a queue of the last 4 chip modes. MCS0 is the latest one, while MCS3 is the oldest one. In addition a qualifier indicates if the FCCU is in the FAULT state when the chip mode has been captured. The chip mode is synchronous to the system clock and provided by a different module while the FCCU state is synchronous to the IRCOSC clock, therefore some uncertainty must be considered regarding the FAULT state indication.

Figure 598. FCCU MCS Register (FCCU_MCS)

Offset: 0x0EC Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VL3	FS3	0	0	MCS3				VL2	FS2	0	0	MCS2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VL1	FS1	0	0	MCS1				VL0	FS0	0	0	MCS0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 570. FCCU_MCS field descriptions

Field	Description
VLx	*Valid* It indicates that the correspondent MCSx and FSx fields are valid. 0: MCSx, FSx fields are not significative 1: MCSx, FSx fields are significative These bits can be read by the software.
FSx	*Fault status* It indicates that the correspondent MCSx field has been captured when the FCCU is in FAULT state. 0: MCSx field captured in any state different from the FAULT state 1: MCSx field captured in FAULT state These bits can be read by the software.
MCSx	*Chip mode* The MCSx is the chip mode. MCS0 = latest state MCS3 = oldest state On any chip mode change the previous chip modes are shifted (MCS3 = MCS2, MCS2= MCS1, MCS1= MCS0) and the latest one is captured in MCS0. These bits can be read by the software.

30.7 Functional description

30.7.1 Definitions

In general, the following definitions are applicable for the fault management:

- HW recoverable fault: the fault indication is a level sensitive signal that is asserted as long as the fault cause has not been removed. Typically the fault signal is latched in a external module at the FCCU. The FCCU state transitions are consequently executed on the state changes of the input fault signal. No SW intervention in the FCCU is required to recover the fault condition.
- SW recoverable fault: the fault indication is a signal asserted without a defined time duration. The fault signal is captured in the FCCU. The fault recovery is executed following a SW recovery procedure (status/flag register clearing).

The following type of reset are applicable (see *8, Reset Generation Module (MC_RGM)*):

- 'Destructive' reset: any type of reset related to a power failure condition that implies a complete system reinitialization

30.7.2 FSM description

The FCCU functionality is depicted by the FSM diagram given in *Figure 599*.

Basically four states are identified with the following meaning:

- CONFIG: the configuration state is used only to modify the default configuration of the FCCU. A sub-set of the FCCU registers, dedicated to define the FCCU configuration (global configuration, reactions to fault, time-out, non-critical fault masking) can be accessed in write mode only in the CONFIG state.
  The CONFIG state is accessible only from NORMAL state and if the configuration is

not locked. The configuration lock can be disabled only by a global reset of the FCCU. The configuration shall always be locked when running a safety critical application. The CONFIG to NORMAL state transition can be executed by SW or automatically following a time-out condition of the watchdog.
The incoming faults, occurring during the configuration phase (CONFIG state) are latched in order to process them when the FCCU is moved into the NORMAL state, according to the selected configuration.

- NORMAL: the FCCU operating state when no faults are occurring. It is also the default state on the reset exit. The FSM will leave the NORMAL state following one of these events:
  - Critical faults → the FCCU moves to the FAULT state
  - Unmasked non-critical faults with the time-out disabled → the FCCU moves to the FAULT state
  - Unmasked non-critical faults with the time-out enabled → the FCCU moves to the ALARM state
  - Masked non-critical faults → the FCCU stays in NORMAL state
- ALARM: the FCCU moves into the ALARM state when an unmasked non-critical fault occurs and the time-out is enabled. The transition to the ALARM state goes along with an interrupt. By definition, this fault may be recovered within a programmable time-out period, before it generates a transition to the FAULT state. The time-out is reinitialized if the FCCU state moves to the NORMAL state. The time-out is stopped if the FCCU state moves to the FAULT state due to a critical-fault occurring when the FCCU is in ALARM state. The time-out restarts following the recovery from the FAULT state.
- FAULT: the FCCU moves into the FAULT state when one of the following condition occurs:
  - Critical fault
  - Time-out related to a non-critical fault when the FCCU is in the ALARM state
  - Unmasked non-critical faults with the time-out disabled
- The transition from NORMAL/ALARM state goes along with the generation of:
  - FCCU_F signalling

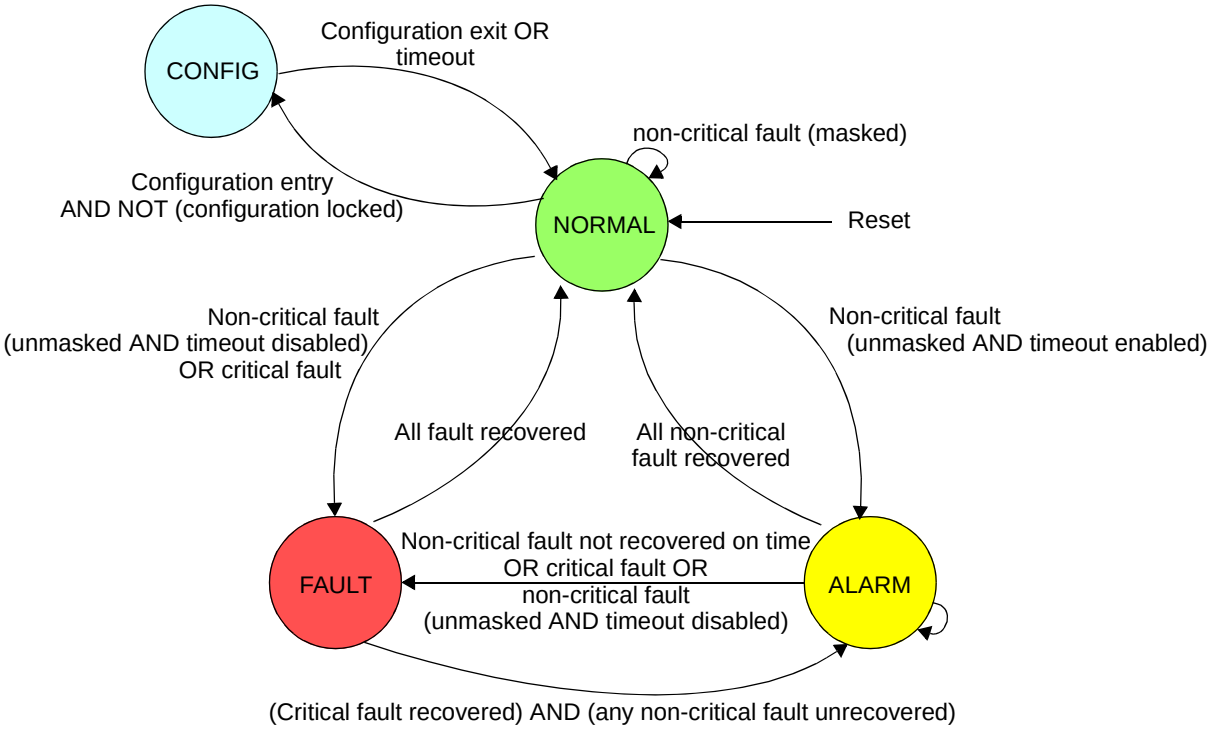


Figure 599. FCCU state diagram

30.7.3 Self-checking capabilities

The FCCU includes some features to support self-checking capabalities of the main FSM in running mode. The FSM unit is duplicated and its state (internal state and relevant outputs) is checked by 2 RCCx units (cycle accurate). The following checks (per IRCOSC clock cycle) are provided by the RCCx units:

- all the FSM outputs and its internal state for the redundant FSM instances are checked to detect runtime faults on the FSM outputs and its internal state
- parity bits (computed at byte level) on the configuration registers are checked to detect run time faults on the configuration inputs of the FSM
- parity bits (computed at byte level) on the interface used to clear the FCCU_CFSx and FCCU_NCFSx status registers
- FCCU_F protocol state in dual-rail, time-switching and bi-stable mode
- common mode signals (handshake interface signals activated only in CONFIG state) congruence with the FSM state

In case of failure, each RCCx unit provides an interrupt request.

The RCCx state is frozen in a status register.

To guarantee the self checking capabilities through the FAULT interface, each critical fault source must be duplicated on a couple of CF inputs.

Two separate IRCOSC clock inputs are routed to the redundant sub-modules (FSMx, RCCx, FCCU_Fx, FAULT-if). An external (to the FCCU) monitor of the IRCOSC clocks and an external reaction ('functional' reset to the MC_RGM) is required to cover potential fault on the IRCOSC clocks.

Self checking capabilities cover the HW reaction of the FCCU due to the assertion of an external fault. The register interface and the handshake modules are not covered by self-checking capabilities. Any operation executed by SW (configuration, fault recovery) must be cross-verified by redundancy checks via SW (registers read following a write/clear operation, internal FCCU state, and so on).

A typical sequence with self checking failure and related reaction is given in *Figure 600*.

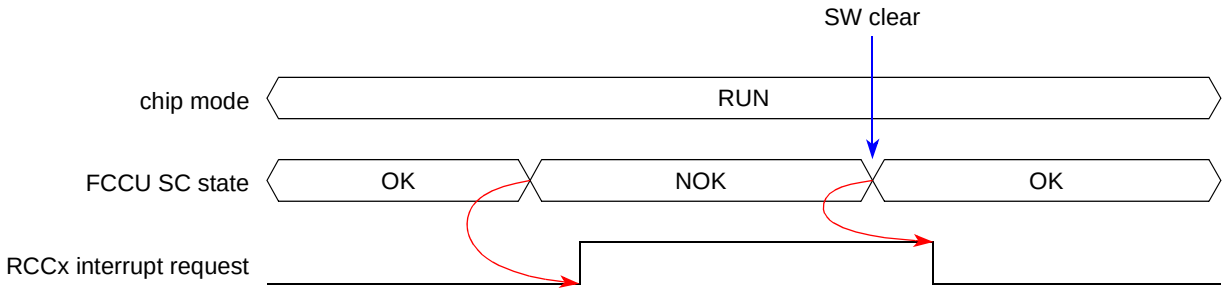


Figure 600. Self-checking reaction

30.7.4 Reset interface

The FCCU has two input resets and two output resets related to the FAULT state.

Table 571. Reset sources

Reset source	Support
External reset	enabled
POR	enabled
STCU	enabled
'Functional' reset	disabled
'Destructive' reset	enabled

30.7.5 Fault priority scheme and nesting

The FAULT state has a higher priority than the ALARM state in case of concurrent fault events (critical and non-critical) that occur in the NORMAL state. In case of concurrent critical faults, the fault reaction corresponds to the worst case (that is, a long 'functional' reset is asserted in case it has been programmed).

The ALARM to FAULT state transition occurs if a critical fault or a non-critical fault (unmasked and with time-out disabled) is asserted in the ALARM state.

Any critical fault (programmed to react with a hard or soft reaction) that occurs when the FCCU is already in the FAULT state causes an immediate hard or soft reaction (long or short 'functional' reset).

The ALARM to NORMAL state transition occurs only if all the non-critical faults (including the faults that have been collected after the entry in the ALARM state) have been cleared (SW or HW recovery) otherwise the FCCU will remain in the ALARM state.

The FAULT to NORMAL state transition occurs only if all the critical and non-critical faults (including the faults that have been collected after the entry in the FAULT/ALARM state) have been cleared (SW or HW recovery) otherwise the FCCU will remain in the FAULT state (if any critical fault is still pending) or will return in the ALARM state (if any non-critical fault is still pending and the time-out is not elapsed).

In general, no fault nesting is supported except for the non-critical versus critical faults that causes a ALARM to FAULT state transition. In this case the NCT timer is stopped until the FAULT state is recovered.

30.7.6 Fault recovery

The following timing diagrams describe the main use cases of the FCCU in terms of fault events and related recovery.

A typical sequence related to a critical FAULT management, given in *Figure 601* or *Figure 602*, is following described:

- Critical fault assertion
- FCCU state transition (automatic): NORMAL → FAULT
- FAULT recovery (by SW): FCCU state transition FAULT → NORMAL

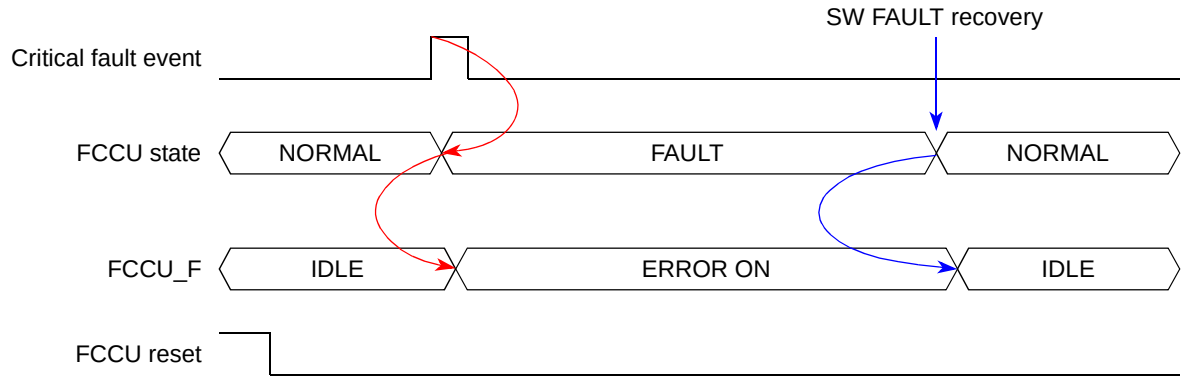


Figure 601. Critical FAULT recovery (a)

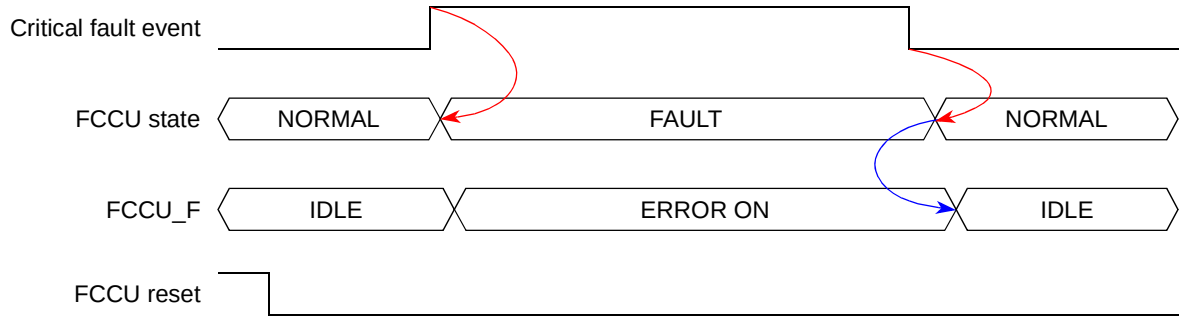


Figure 602. Critical FAULT recovery (b)

A typical sequence related to a non-critical FAULT management (ALARM state), given in *Figure 603* and *Figure 604*, is following described:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - alarm interrupt request
  - time-out running
- Chip mode: RUN
- Alarm interrupt management
  - FAULT recovery (by SW): FCCU state transition ALARM → NORMAL

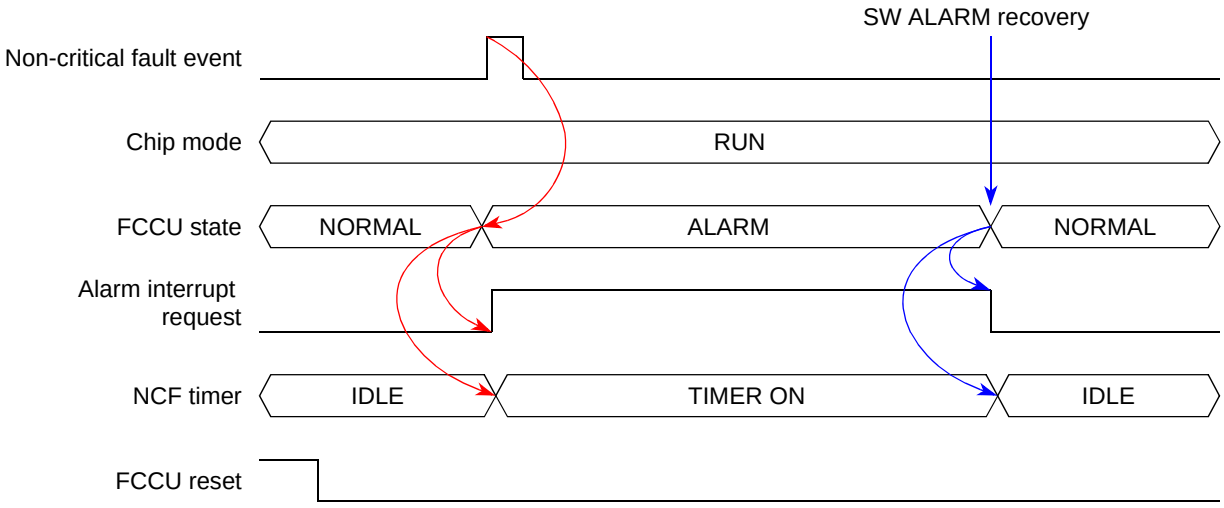


Figure 603. Non-critical FAULT (ALARM state) recovery (a)

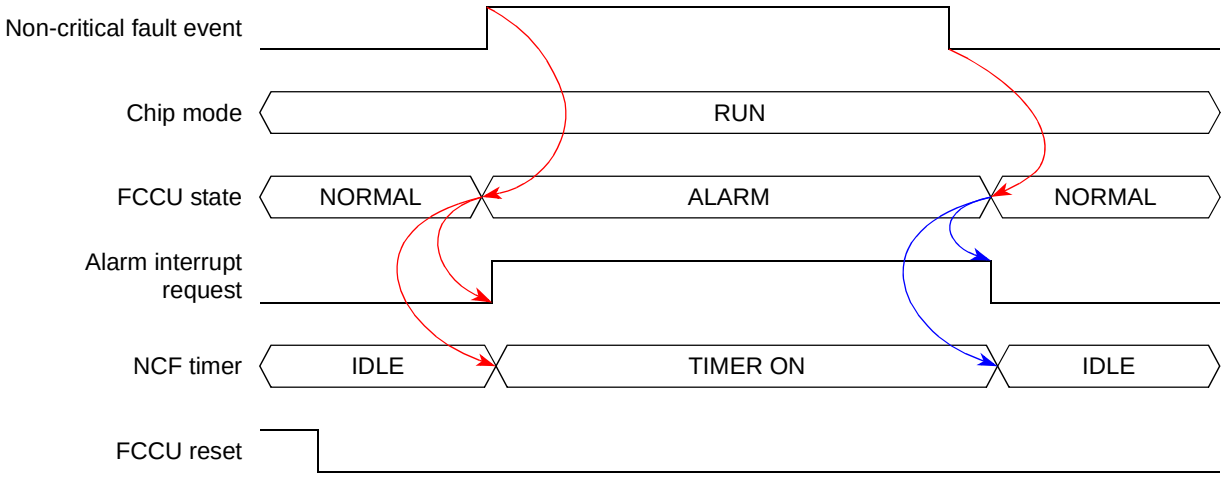


Figure 604. Non-critical FAULT (ALARM state) recovery (b)

A typical sequence related to a non-critical FAULT management (ALARM → FAULT state), given in *Figure 605*, is following described:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Time-out running
- FCCU state transition (following the time-out trigger): ALARM → FAULT

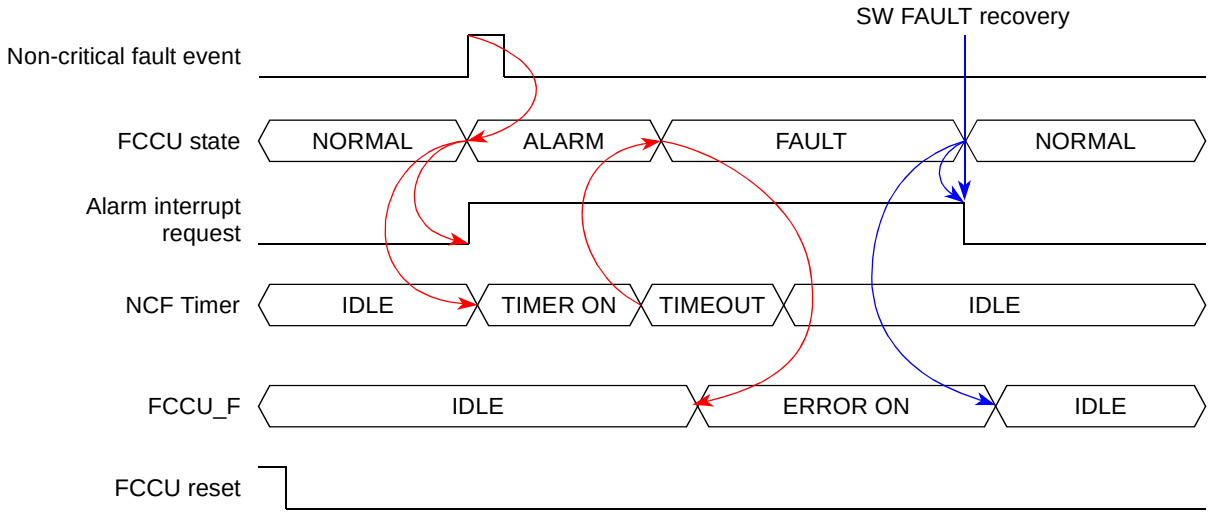


Figure 605. Non-critical FAULT (ALARM -> FAULT state) recovery

A typical sequence related to a critical FAULT (with nesting) management, given in *Figure 606*, is following described:

- Critical fault assertion
- FCCU state transition (automatic): NORMAL → FAULT
- FAULT recovery (by SW): FCCU state transition FAULT → NORMAL

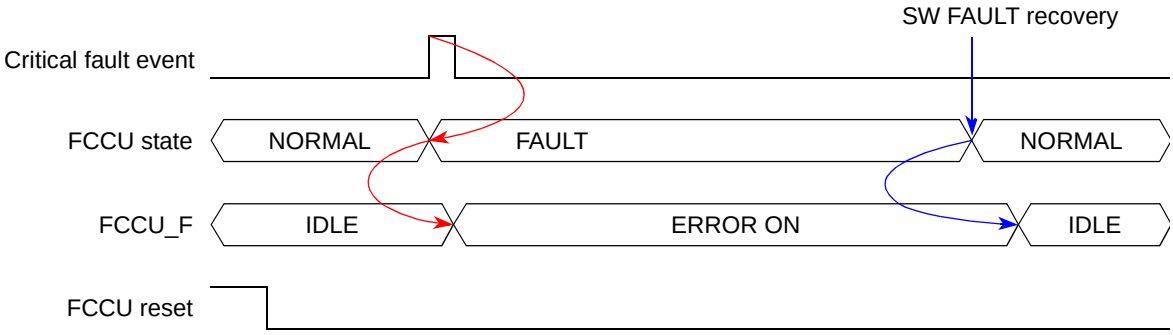


Figure 606. Critical FAULT (nesting) recovery

A typical sequence related to a critical FAULT (with non-critical fault nesting) management (ALARM → FAULT → ALARM state), given in *Figure 607*, where the faults are recovered sequentially, is following described:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Time-out running
- Critical fault assertion
- FCCU state transition (automatic): ALARM → FAULT
- FAULT (CF) recovery (by SW): FCCU state transition FAULT → ALARM, because only the critical fault has been recovered
- Time-out is still running
- Alarm interrupt management
  - FAULT (NCF) recovery (by SW): FCCU state transition ALARM → NORMAL

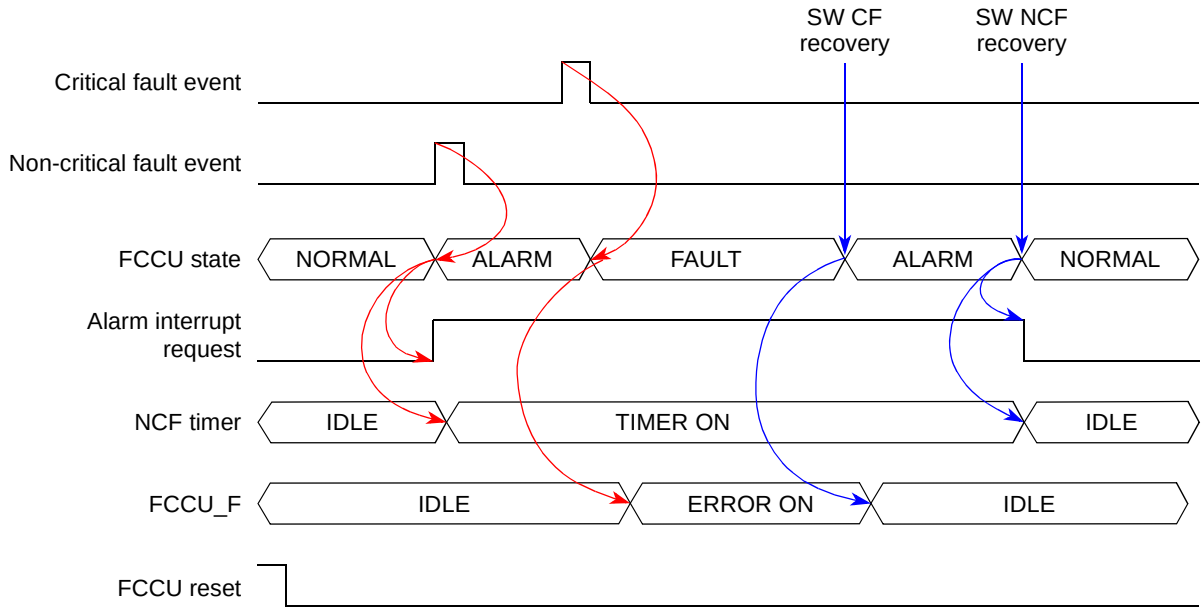


Figure 607. Critical FAULT (and non-critical FAULT nesting) recovery

30.7.7 NVM interface

The NVM provides the FCCU with the initial configuration information shown in *Table 572*.

Table 572. NVM configuration

Flash memory option bit locations	Description
BIU4[20]	Initial FCCU_CFG.CM value
BIU4[21]	Initial FCCU_CFG.SM value
BIU4[22]	Initial FCCU_CFG.PS value
BIU4[23:25]	Initial FCCU_CFG.FOM value
BIU4[26:31]	Initial FCCU_CFG.FOP value

Figure 608 shows the FCCU configuration sequence after a power-on, ‘destructive’, or external reset.

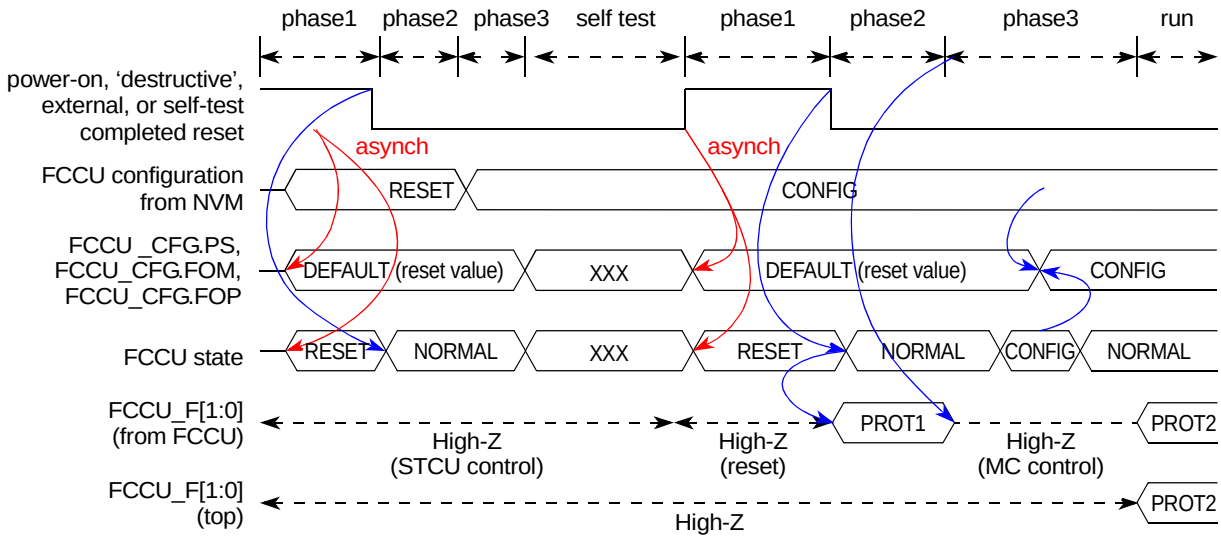


Figure 608. NVM interface

30.7.8 FCCU_F interface

The FCCU provides 2 external bidirectional signals (FCCU_F interface). Different protocols for the FCCU_F interface are supported, selecting the FCCU_CFG.FOM register field:

- Dual rail protocol
- Time switching protocol
- Bi-stable protocol
- Test mode

The signal polarity and the frequency can be programmed, setting the FCCU_CFG.PS and FCCU_CFG.FOP register fields. All the diagrams and tables are related to the default configuration selection (FCCU_CFG.FOP = 0b), switching mode (FCCU_CFG.SM = 0b) and config mode (FCCU_CFG.CM = 0b). In case of inverted polarity (FCCU_CFG.FOP = 1b) all the values on the FCCU_F output pins are inverted.

2 modes can be programmed to define the FCCU_F protocol transitions in dual-rail or time-switching mode:

- Slow switching mode: no FCCU_F frequency violation during the FCCU state transition (NORMAL to ERROR or viceversa and CONFIG to NORMAL). The FCCU_F protocol transition occurs after a max delay equal to the duration of the semi-period of the FCCU_F frequency.
- Fast switching mode: The FCCU_F protocol transition (NORMAL to ERROR or viceversa and CONFIG to NORMAL) occurs immediately. A pulse with the minimum duration corresponding to 16 MHz / 1024 (IRCOSC clock) period can occurs in fast switching mode. It implies a frequency violation of the FCCU_F protocol.

Two modes, depending on the FCCU_CFG.CM bit setting, can be programmed to define the FCCU_F protocol in CONFIG state:

- configuration labelling: the CONFIG state is marked by a specific FCCU_F setting
- configuration transparency: the CONFIG and NORMAL state are equivalent

The FCCU_F frequency is programmable based on the IRCOSC clock frequency divided by a fixed prescaler (1024).

The external monitor of the FCCU_F protocol should oversample the FCCU_F signals in order to synchronize periodically the external clock (used by the monitor) and the IRCOSC clock detecting the edge transition of the FCCU_F protocol in dual-rail or time-switching mode.

Note: The initial values, after the reset phase, of the FCCU_CFG.CM, FCCU_CFG.SM, FCCU_CFG.PS, FCCU_CFG.FOP, FCCU_CFG.FOM registers are set by the NVM interface (see Section 30.7.7, NVM interface).

Dual-rail protocol

Dual-rail encoding is an alternate method for encoding bits. In contrast with classical encoding, where each signal carries a single-bit value, dual-rail encoded circuits use two wires to carry each bit. The encoding scheme is given in *Table 573* and the related timing diagram is given in *Figure 609* and *Figure 610*.

Table 573. Dual-rail encoding

Logical state	Dual-rail encoding (output pins FCCU_F[1:0])	Note
non-faulty	10	toggling
non-faulty	01	
faulty	00	toggling
faulty	11	
reset	high-Z	no toggling
configuration	high-Z	when FCCU_CFG.CM = 0
	= non-faulty	when FCCU_CFG.CM = 1

As long as FCCU is in NORMAL or ALARM state, output will show “no-faulty“signal. Output pins FCCU_F[0] and FCCU_F[1] will toggle between 01 and 10 with a given frequency. By default the frequency is the IRCOSC clock frequency divided by 18*1024.

During the RESET phase and during self testing the output pins are set as “high impedance”.

Note: Figure 609 and Figure 610 are formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

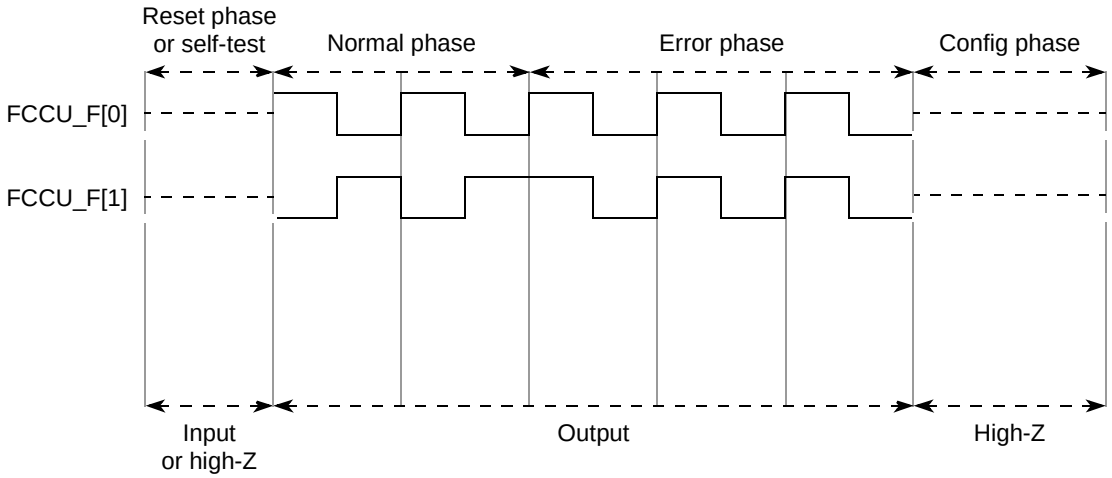


Figure 609. Dual-rail protocol (slow switching mode)

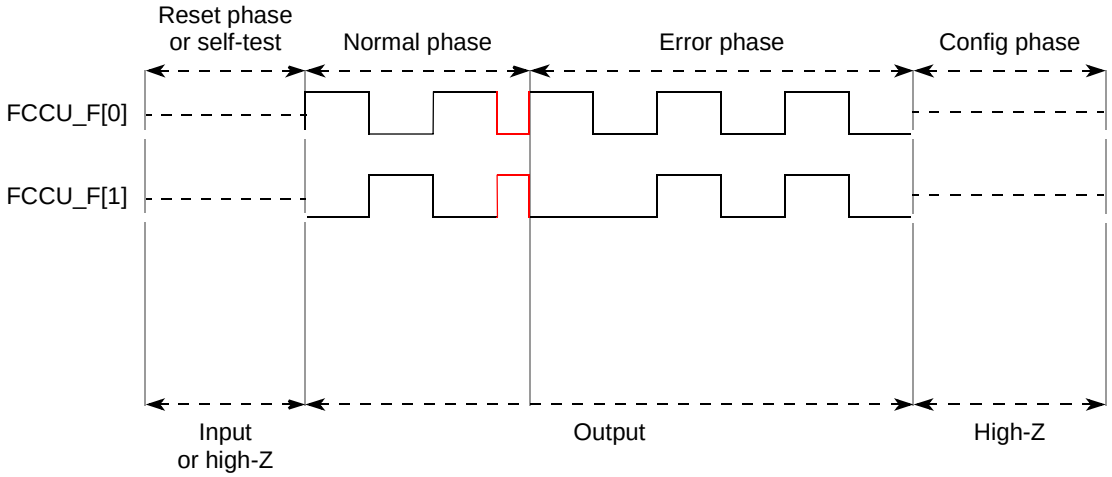


Figure 610. Dual-rail protocol (fast switching mode)

Time switching protocol

The encoding scheme is given in *Table 574* and the related timing diagram is given in *Figure 611*.

Table 574. Time switching encoding

Logical state	Time switching encoding (output pins FCCU_F[1:0])	Note
non-faulty	10	toggling
non-faulty	01	
faulty	10	no toggling
reset	high-Z	no toggling

Table 574. Time switching encoding

Logical state	Time switching encoding (output pins FCCU_F[1:0])	Note
configuration	01	when FCCU_CFG.CM = 0
	= non-faulty	when FCCU_CFG.CM = 1

As long as FCCU is in NORMAL or ALARM state, outputs will show “non-faulty“ signal. Output pins #0, #1 will toggle between “01” and “10” with a given frequency. By default the frequency is the IRCOSC clock frequency divided by 18*1024.

In the FAULT state, the output pin FCCU_F[0] is set as low.

In Time Switching mode the second output (FCCU_F[1]) is the inverted signal of first output (FCCU_F[0]). Values 00 on the outputs indicate a fault in the error out protocol itself. This state must be considered as critical fault, because no reliable error out indication is available any more.

In the RESET phase the output pins are set as “high impedance”.

Note: Figure 611 is formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

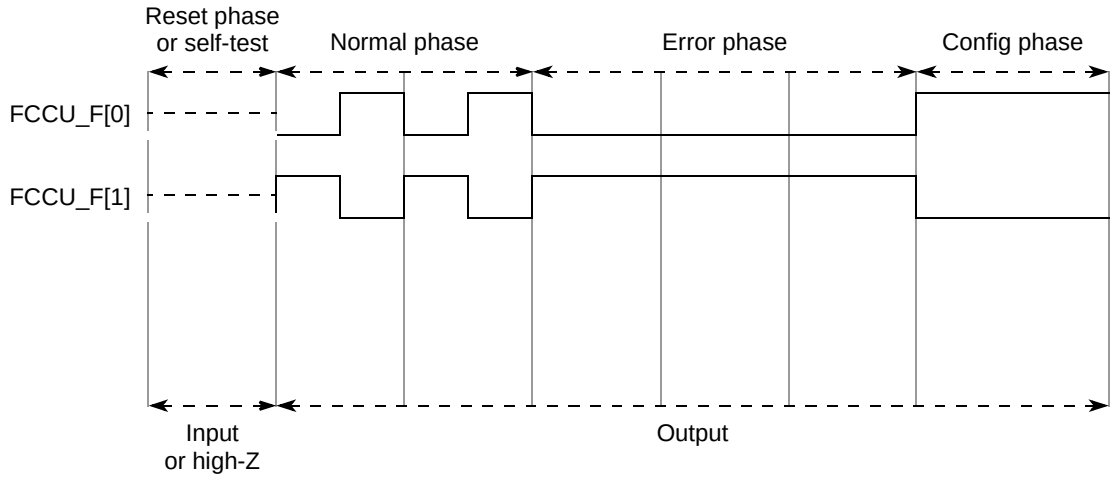


Figure 611. Time-switching protocol

Bi-stable protocol

The encoding scheme is given in *Table 575* and the related timing diagram is given in *Figure 612*.

Table 575. Bi-stable encoding

Logical state	Bi-stable encoding (output pins FCCU_F[1:0])	Note
non-faulty	01	no toggling
faulty	10	no toggling
reset	high-Z	no toggling
configuration	10	when FCCU_CFG.CM = 0
	= non-faulty	when FCCU_CFG.CM = 1

In the FAULT state, the faulty logical state is indicated. In NORMAL or ALARM state, “no-faulty” state is indicated. In Bi-stable mode the second output (FCCU_F[1]) is the inverted signal of first output (FCCU_F[0]).

In the RESET phase the output pins are set as “high impedance”.

Note: Figure 612 is formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

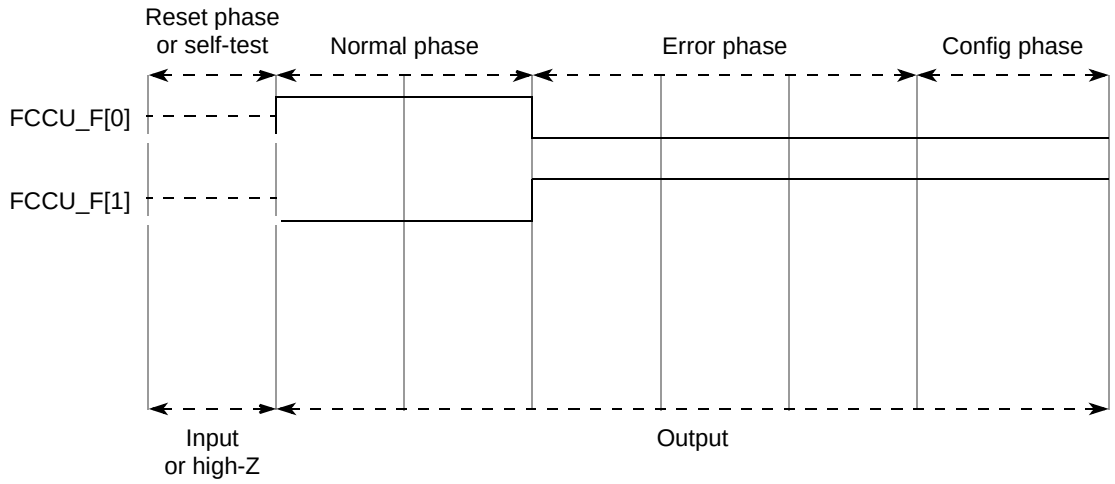


Figure 612. Bi-stable protocol

30.7.9 Fault mapping

Table 576 and *Table 577* show the source of the fault signals and the type of fault input these signals are connected to at the FCCU.

Table 576. FCCU mapping of critical faults

Critical fault	Module	Fault type
CF[0]	CORE_0	Core Checkstop mode entered
CF[1]	CORE_0	Core reset output
CF[2]	CORE_1	Core Checkstop mode entered

Table 576. FCCU mapping of critical faults (continued)

Critical fault	Module	Fault type
CF[3]	CORE_1	Core reset output
CF[4]	FLASH	Flash fatal error
CF[5]	JTAG	JTAG reset (TAP controller)
CF[6]	Code Flash	ECC Multi bit error
CF[7]	Data Flash	ECC Multi bit error
CF[8]	SRAM	ECC Multi bit error
CF[14]	SWT_0	Software Watchdog Timer Reset
CF[15]	SWT_1	Software Watchdog Timer Reset
CF[16]	ECSM_0	ECC not correctable error
CF[17]	ECSM_1	ECC not correctable error
CF[22]	SSCM_XFER_ERR	SSCM transfer error

Table 577. FCCU mapping of non-critical faults

Non-critical fault	Module	Fault type
NCF[2]	FMPLL_0	Loss of lock
NCF[4]	CMU_0	Loss of XOSC clock
NCF[5]	CMU_0	FMPLL_0_CLK frequency out of range
NCF[6]	CMU_1	SYS_CLK frequency out of range
NCF[8]	ECSM_0	ECC 1-bit error correction notification
NCF[9]	ECSM_1	ECC 1-bit error correction notification
NCF[10]	CRC_0	CRC Output Check
NCF[11]	CRC_1	CRC Output Check
NCF[13]	PMU	LVD 1.2 digital Fault
NCF[14]	PMU	LVD 4.5 Fault
NCF[15]	PMU	LVD 2.7 VREG Fault
NCF[16]	PMU	LVD 2.7 FLASH Fault
NCF[17]	PMU	LVD 2.7 I/O Fault
NCF[21]	MC_ME	Software device reset

31 Wakeup Unit (WKPU)

31.1 Overview

The Wakeup Unit (WKPU) supports one external source that causes non-maskable interrupt requests.

31.2 Features

The WKPU provides non-maskable interrupt support with these features:

- 1 NMI source
- 1 analog glitch filter
- Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
- Edge detection

31.3 External signal description

The WKPU has one signal input that can be used as non-maskable interrupt.

Note: The user should be aware that the Wake-up pins are enabled in ALL modes, therefore, the Wake-up pins should be correctly terminated to ensure minimal current consumption. Any unused Wake-up signal input should be terminated by using an external pull-up or pull-down, or by internal pull-up enabled at WKUP_WIPUER. Also care has to be taken on packages where the Wake-up signal inputs are not bonded. For these packages the user must ensure the internal pull-up are enabled for those signals not bonded.

31.4 Memory map and registers description

This section provides a detailed description of all registers accessible in the WKPU module.

31.4.1 Memory map

Table 578 lists the WKPU registers.

Table 578. WKPU memory map

Offset from WKPU_BASE (0xC3F9_4000)	Register	Location
0x0000	NSR—NMI Status Flag Register	*on page 992*
0x0004–0x0007	Reserved	
0x0008	NCR—NMI Configuration Register	*on page 993*
0x000C–0x3FFF	Reserved	

31.4.2 Registers description

This section describes the Wakeup Unit registers.

NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Figure 613. NMI Status Flag Register (NSR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NIF	NOVF	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 579. NSR field descriptions

Field	Description
0 NIF	NMI Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NCR.NREEor NCR.NFEE is set), NIFcauses an interrupt request. 0: No event has occurred on the pad. 1: An event as defined by NRRC.NREE or NCR.NFEE has occurred.
1 NOVF	NMI Overrun Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. It will be a copy of the current NIF value whenever an NMI event occurs, thereby indicating to the software that an NMI occurred while the last one was not yet serviced. If enabled (NCR.NREEor NCR.NFEE set), NOVF causes an interrupt request. 0: No overrun has occurred on NMI input. 1: An overrun has occurred on NMI input.

NMI Configuration Register (NCR)

This register holds the configuration bits for the non-maskable interrupt settings.

Figure 614. NMI Configuration Register (NCR)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NLOCK	NDSS		0	0	NREE	NFEE	NFE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 580. NCR field descriptions

Field	Description
0 NLOCK	NMI Configuration Lock Register Writing a 1 to this bit locks the configuration for the NMI until it is unlocked by a system reset. Writing a 0 has no effect.
1-2 NDSS	NMI Destination Source Select 00: Non-maskable interrupt 01: Critical interrupt 10: Machine check request 11: Reserved
5 NREE	NMI Rising-edge Events Enable 0: Rising-edge event is disabled 1: Rising-edge event is enabled
6 NFEE	NMI Falling-edge Events Enable 0: Falling-edge event is disabled 1: Falling-edge event is enabled
7 NFE	NMI Filter Enable Enable analog glitch filter on the NMI pad input. 0: Filter is disabled 1: Filter is enabled

Note: Writing a 0 to both NREEand NFEE disables the NMI functionality completely (that is, no system wakeup or interrupt will be generated on any pad activity)!

31.5 Functional description

31.5.1 General

This section provides a complete functional description of the Wakeup Unit.

31.5.2 Non-Maskable Interrupts

The Wakeup Unit supports one non-maskable interrupt, which is allocated to pin 1.

The Wakeup Unit supports the generation of three types of interrupts from the NMI input to the device. The Wakeup Unit supports the capturing of a second event per NMI input before the interrupt is cleared, thus reducing the chance of losing an NMI event.

Each NMI passes through a bypassable analog glitch filter.

Note: Glitch filter control and pad configuration should be done while the NMI is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

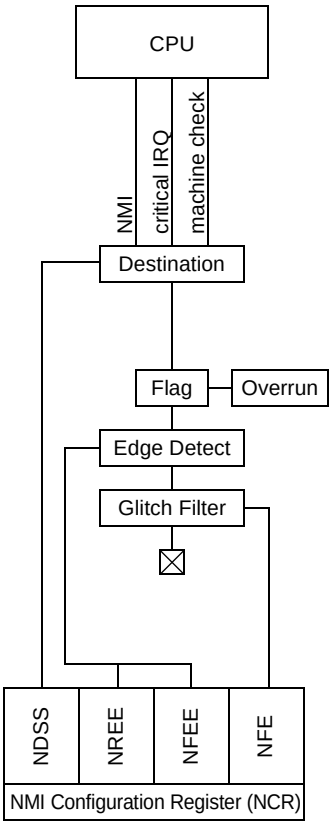


Figure 615. NMI pad diagram

NMI management

The NMI can be enabled or disabled using the single NCR register laid out to contain all configuration bits for an NMI in a single byte (see *Figure 614*). The pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled results in no interrupt being detected and should not be configured.

The active NMI edge is controlled by the user through the configuration of the NREE and NFEE bits.

Note: After reset, NREE and NFEE are set to 0, therefore the NMI functionality is disabled after reset and must be enabled explicitly by software.

Once the pad's NMI functionality has been enabled, the pad cannot be reconfigured in the IOMUX to override or disable the NMI.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS bits. See *Table 580* for details.

An NMI supports a status flag and an overrun flag, which are located in the NSR register (see *Figure 613*). This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register. The status flag is set whenever an NMI event is detected. The overrun flag is set whenever an NMI event is detected and the status flag is set (that is, has not yet been cleared).

Note: The overrun flag is cleared by writing a 1 to the appropriate overrun bit in the NSR register. If the status bit is cleared and the overrun bit is still set, the pending interrupt will not be cleared.

32 Periodic Interrupt Timer (PIT)

32.1 Introduction

The Periodic Interrupt Timer (PIT) block implements several timers that can be used for DMA triggering, general purpose interrupts and system wakeup.

Figure 616 shows the PIT block diagram.

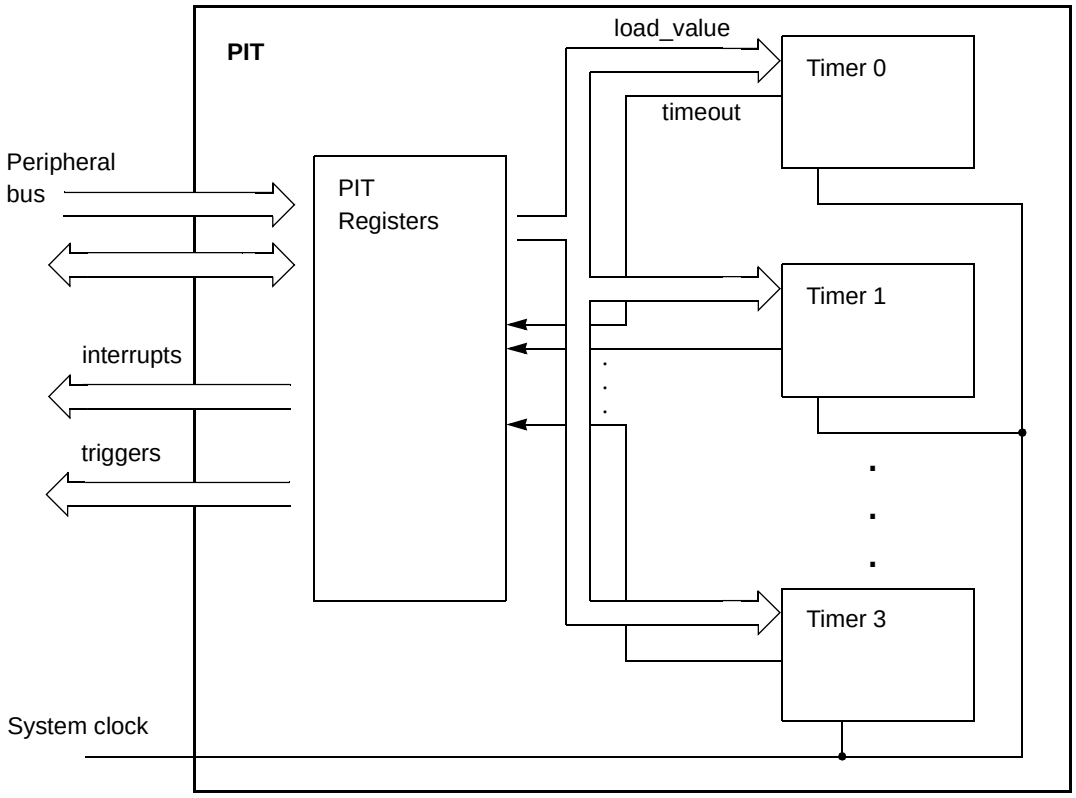


Figure 616. PIT block diagram

32.1.1 Overview

This chapter describes the function of the Periodic Interrupt Timer block (PIT). The PIT is an array of four timers that can be used to raise interrupts and trigger DMA channels.

32.1.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses to initiate DMA transfers with other peripherals (ex: initiate a SPI message transfer sequence)
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

32.2 Signal description

The PIT module has no external pins.

32.3 Memory map and registers description

This section provides a detailed description of all registers accessible in the PIT module.

32.3.1 Memory map

Table 581 gives an overview on all PIT registers.

Table 581. PIT memory map

Offset from PIT_BASE (0xC3FF_0000)	Register	Location
0x0000	*PITMCR—PIT Module Control Register*	*on page 998*
0x0004–0x00FF	Reserved	
Timer Channel 0		
0x0100	LDVAL0—Timer 0 Load Value Register	*on page 999*
0x0104	CVAL0—Timer 0 Current Value Register	*on page 1000*
0x0108	TCTRL0—Timer 0 Control Register	*on page 1001*
0x010C	*TFLG0—Timer 0 Flag Register*	*on page 1002*
Timer Channel 1		
0x0110	LDVAL1—Timer 1 Load Value Register	*on page 999*
0x0114	CVAL1—Timer 1 Current Value Register	*on page 1000*
0x0118	TCTRL1—Timer 1 Control Register	*on page 1001*
0x011C	*TFLG1—Timer 1 Flag Register*	*on page 1002*
Timer Channel 2		
0x0120	LDVAL2—Timer 2 Load Value Register	*on page 999*
0x0124	CVAL2—Timer 2 Current Value Register	*on page 1000*
0x0128	TCTRL2—Timer 2 Control Register	*on page 1001*
0x012C	*TFLG2—Timer 2 Flag Register*	*on page 1002*
Timer Channel 3		
0x0130	LDVAL3—Timer 3 Load Value Register	*on page 999*
0x0134	CVAL3—Timer 3 Current Value Register	*on page 1000*
0x0138	TCTRL3—Timer 3 Control Register	*on page 1001*
0x013C	*TFLG3—Timer 3 Flag Register*	*on page 1002*
0x0140–0x3FFF	Reserved	

Note: Reserved registers read as 0. Writes have no effect.

32.3.2 Registers description

This section describes in address order all the PIT registers and their individual bits. PIT registers are accessible only when the core is in supervisor mode.

PIT Module Control Register (PITMCR)

This register controls whether the timer clocks are enabled and whether the timers run in debug mode.

Figure 617. PIT Module Control Register (PITMCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 582. PITMCR field descriptions

Field	Description
MDIS	Module Disable Used to disable the module clock. This bit should be enabled before any other setup is done. 0: Clock for PIT Timers is enabled 1: Clock for PIT Timers is disabled (default)
FRZ	Freeze Allows the timers to be stopped when the device enters debug mode. 0: Timers continue to run in debug mode. 1: Timers are stopped in debug mode.

Timer Load Value Register *n* (LDVAL*n*)

These registers select the timeout period for the timer interrupts.

Figure 618. Timer Load Value Register *n* (LDVAL*n*)

Channel Base + 0x0000
LDVAL0 = PIT_BASE + 0x0100
Address: LDVAL1 = PIT_BASE + 0x0110 Access: User read/write
LDVAL2 = PIT_BASE + 0x0120
LDVAL3 = PIT_BASE + 0x0130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TSV 31	TSV 30	TSV 29	TSV 28	TSV 27	TSV 26	TSV 25	TSV 24	TSV 23	TSV 22	TSV 21	TSV 20	TSV 19	TSV 18	TSV 17	TSV 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TSV 15	TSV 14	TSV 13	TSV 12	TSV 11	TSV 10	TSV 9	TSV 8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 583. LDVAL*n* field descriptions

Field	Description
TSV*n*	Time Start Value Bits These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see *Figure 623*).

Current Timer Value Register *n* (CVAL*n*)

These registers indicate the current timer position.

Figure 619. Current Timer Value register *n* (CVAL*n*)

Channel Base + 0x0004
CVAL0 = PIT_BASE + 0x0104
Address: CVAL1 = PIT_BASE + 0x0114 Access: User read-only
CVAL2 = PIT_BASE + 0x0124
CVAL3 = PIT_BASE + 0x0134

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 584. CVAL*n* field descriptions

Field	Description
TVL*n*	Current Timer Value These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see *Figure 617*).

Timer Control Register *n* (TCTRL*n*)

The TCTRL register contains the control bits for each timer.

Figure 620. Timer Control register *n* (TCTRL*n*)

Channel Base + 0x0008
TCTRL0 = PIT_BASE + 0x0108
Address: TCTRL1 = PIT_BASE + 0x0118 Access: User read/write
TCTRL2 = PIT_BASE + 0x0128
TCTRL3 = PIT_BASE + 0x0138

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 585. TCTRL*n* field descriptions

Field	Description
TIE	Timer Interrupt Enable Bit 0: Interrupt requests from Timer x are disabled 1: Interrupt will be requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit 0: Timer will be disabled 1: Timer will be active

Timer Flag Register *n* (TFLG*n*)

These registers contain the PIT interrupt flags.

Figure 621. Timer Flag register *n* (TFLG*n*)

Channel Base + 0x000C
TFLG0 = PIT_BASE + 0x010C
Address: TFLG1 = PIT_BASE + 0x011C Access: User read/write
TFLG2 = PIT_BASE + 0x012C
TFLG3 = PIT_BASE + 0x013C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 586. TFLG*n* field descriptions

Field	Description
TIF	Time Interrupt Flag TIF is set to 1 at the end of the timer period.This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0: Time-out has not yet occurred 1: Time-out has occurred

32.4 Functional description

32.4.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line.

Timers

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL registers, then count down until they reach 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse, and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL registers.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see *Figure 622*).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see *Figure 623*).

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value will then be loaded after the next trigger event (see *Figure 624*).

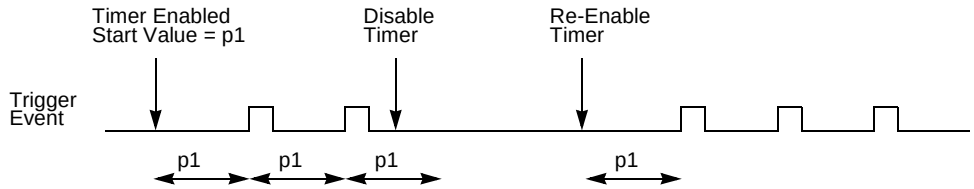


Figure 622. Stopping and starting a timer

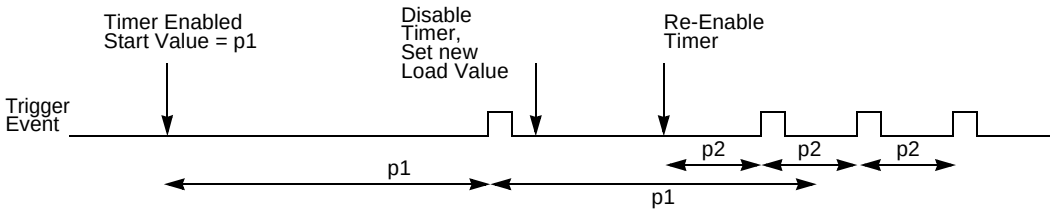


Figure 623. Modifying running timer period

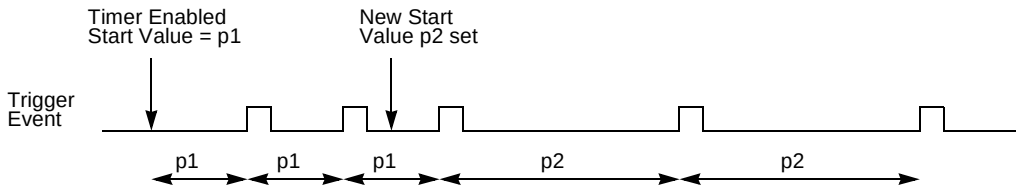


Figure 624. Dynamically setting a new load value

Debug mode

In Debug mode, the timers are frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g., timer values) and then continue the operation.

32.4.2 Interrupts

All of the timers support interrupt generation. Refer to *Chapter 9: Interrupt Controller (INTC)* for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

32.5 Initialization and application information

32.5.1 Example configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz
- Timer 1 creates an interrupt every 5.12 ms
- Timer 3 creates a trigger event every 30 ms

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITMCR.

The 50 MHz clock frequency equates to a clock period of 20 ns. Timer 1 needs to trigger every 5.12 ms/20 ns = 256000 cycles and timer 3 every 30 ms/20 ns = 1500000 cycles. The value for the LDVAL register trigger would be calculated as (period / clock period) – 1.

The LDVAL registers must be configured as follows:

- LDVAL for Timer 1: 0x0003_E7FF
- LDVAL for Timer 3: 0x0016_E35F

The interrupt for Timer 1 is enabled by setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register, bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL = PIT_TIE; // let RTI generate interrupts
PIT_RTI_TCTRL |= PIT_TEN; // start RTI

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

33 System Timer Module (STM)

33.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

33.2 Features

The STM has the following features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

33.3 Modes of operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

33.4 External signal description

The STM does not have any external interface signals.

33.5 Memory map and registers description

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination. STM registers are accessible only when the core is in supervisor mode.

33.5.1 Memory map

The STM memory map is shown in *Table 587*.

Table 587. STM memory map

Offset from STM_BASE 0xFFF3_C000 (STM_0) 0x8FF3_C000 (STM_1)	Register	Location
0x0000	STM_CR—STM Control Register	*on page 1006*
0x0004	STM_CNT—STM Counter Value	*on page 1007*
0x0008–0x000F	Reserved	
0x0010	STM_CCR0—STM Channel 0 Control Register	*on page 1008*
0x0014	STM_CIR0—STM Channel 0 Interrupt Register	*on page 1008*
0x0018	STM_CMP0—STM Channel 0 Compare Register	*on page 1009*
0x001C	Reserved	
0x0020	STM_CCR1—STM Channel 1 Control Register	*on page 1008*
0x0024	STM_CIR1—STM Channel 1 Interrupt Register	*on page 1008*
0x0028	STM_CMP1—STM Channel 1 Compare Register	*on page 1009*
0x002C	Reserved	
0x0030	STM_CCR2—STM Channel 2 Control Register	*on page 1008*
0x0034	STM_CIR2—STM Channel 2 Interrupt Register	*on page 1008*
0x0038	STM_CMP2—STM Channel 2 Compare Register	*on page 1009*
0x003C	Reserved	
0x0040	STM_CCR3—STM Channel 3 Control Register	*on page 1008*
0x0044	STM_CIR3—STM Channel 3 Interrupt Register	*on page 1008*
0x0048	STM_CMP3—STM Channel 3 Compare Register	*on page 1009*
0x004C–0x3FFF	Reserved	

33.5.2 Registers description

The following sections detail the individual registers within the STM programming model.

STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

Figure 625. STM Control Register (STM_CR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CPS[7:0]								0	0	0	0	0	0	FRZ	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 588. STM_CR field descriptions

Field	Description
CPS[7:0]	Counter Prescaler Selects the clock divide value for the prescaler (1 - 256). 0x00 Divide system clock by 1. 0x01 Divide system clock by 2. ... 0xFF Divide system clock by 256.
FRZ	Freeze Allows the timer counter to be stopped when the device enters debug mode. 0 STM counter continues to run in debug mode. 1 STM counter is stopped in debug mode.
TEN	Timer Counter Enabled 0 Counter is disabled. 1 Counter is enabled.

STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Figure 626. STM Count Register (STM_CNT)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 589. STM_CNT field descriptions

Field	Description
CNT	Timer count value used as the time base for all channels. When enabled, the counter increments at the rate of the system clock divided by the prescale value.

STM Channel Control Register (STM_CCR*n*)

The STM Channel Control Register (STM_CCR*n*) enables and services channel *n* of the timer.

Figure 627. STM Channel Control Register (STM_CCR*n*)

Address: Base + 0x0010 (STM_CCR0)
Base + 0x0020 (STM_CCR1)
Base + 0x0030 (STM_CCR2)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 590. STM_CCR*n* field descriptions

Field	Description
CEN	Channel Enable 0 The channel is disabled. 1 The channel is enabled.

STM Channel Interrupt Register (STM_CIR*n*)

The STM Channel Interrupt Register (STM_CIR*n*) enables and services channel *n* of the timer.

Figure 628. STM Channel Interrupt Register (STM_CIR*n*)

Address: Base + 0x0014 (STM_CIR0) / Base + 0x0024 (STM_CIR1) / Base + 0x0034 (STM_CIR2) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 591. STM_CIR*n* field descriptions

Field	Description
CIF	Channel Interrupt Flag The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to a match on the channel.

STM Channel Compare Register (STM_CMP*n*)

The STM Channel Compare Register (STM_CMP*n*) holds the compare value for channel *n*.

Figure 629. STM Channel Compare Register (STM_CMP*n*)

Address: Base + 0x0018 (STM_CMP0) / Base + 0x0028 (STM_CMP1) / Base + 0x0038 (STM_CMP2) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CMP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CMP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 592. STM_CMP*n* field descriptions

Field	Description
CMP	Compare value for channel n If the STM_CCR*n*[CEN] bit is set and the STM_CMP*n* register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIR*n*[CIF] bit is set.

33.6 Functional description

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM_CR[FRZ] bit. When the STM_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at 0xFFFF_FFFF to 0x0000_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM_CCR*n*), a channel interrupt register (STM_CIR*n*) and a channel compare register (STM_CMP*n*). The channel is enabled by setting the STM_CCR*n*[CEN] bit. When enabled, the channel will set the STM_CIR[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM_CIR*n*[CIF] bit. A write of 0 to the STM_CIR*n*[CIF] bit has no effect.

34 Cyclic Redundancy Check (CRC)

34.1 Introduction

The Cyclic Redundancy Check (CRC) computing unit is dedicated to the computation of CRC, thus off-loading the CPU. The SPC56xP60/54 CRC supports three contexts. Each context has a separate CRC computation engine in order to allow the concurrent computation of the CRC of multiple data streams. The CRC computation is performed at speed without wait states insertion. Bit-swap and bit-inversion operations can be applied on the final CRC signature. Each context can be configured with one of three hard-wired polynomials, normally used for most of the standard communication protocols. The data stream supports multiple data width (byte/half-word/word) formats.

34.1.1 Glossary

- CRC: cyclic redundancy check
- CPU: central processing unit
- DMA: direct memory access
- CCITT: ITU-T (for Telecommunication Standardization Sector of the International Telecommunications Union)
- SW: software
- WS: wait state
- SPI: serial peripheral interface

34.2 Main features

- 3 contexts for the concurrent CRC computation
- Separate CRC engine for each context
- Zero-wait states during the CRC computation (pipeline scheme)
- 3 hard-wired polynomials (CRC-8 VDA CAN, CRC-16-CCITT and CRC-32 ethernet)
- Support for byte/half-word/word width of the input data stream

34.2.1 Standard features

- IPS bus interface
- CRC-8 VDA CAN
- CRC-16-CCITT
- CRC-32 ethernet

34.3 Block diagram

Figure 630 shows the top level diagram of the CRC unit.

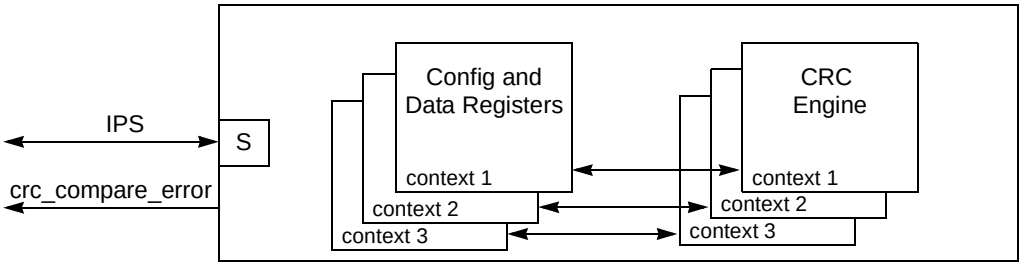


Figure 630. CRC top level diagram

34.3.1 IPS bus interface

The IPS bus interface is a slave bus used for configuration and data streaming (CRC computation) purposes via CPU or DMA. The following bus operations (contiguous byte enables) are supported:

- Word (32-bit) data write/read operations to any registers
- Low and high half-words (16-bit, data[31:16] or data[15:0]) data write/read operations to any registers
- Byte (8-bit, data[31:24] or data[23:16] or data[15:8] or data[7:0]) data write/read operations to any registers
- Any other operation (free byte enables or other operations) must be avoided.

The CRC generates a transfer error in the following cases:

- Any write/read access to the register addresses not mapped on the peripheral but included in the address space of the peripheral.
- Any write/read operation different from byte/hword/word (free byte enables or other operations) on each register.

The registers of the CRC module are accessible (read/write) in each access mode: user, supervisor, or test.

In terms of bus performance of the operations, following the summary:

- 0 WS (single bus cycle) for each write/read operations to the CRC_CFG and CRC_INP registers
- 0 WS (single bus cycle) for each write operation to the CRC_ CSTAT register
- Double WS (3 bus cycles) for each read operation to the CRC_ CSTAT or CRC_OUTP registers immediately following (next clock cycle) a write operation to the CRC_CSTAT, CRC_INP or CRC_CFG registers belonging to the same context. In all the other cases no WS are inserted.

34.4 Functional description

The CRC module supports the CRC computation for each context. Each context has a own complete set of registers including the CRC engine. The data flow of each context can be interleaved. The data stream can be structured as a sequence of byte, half-words or words. The input data sequence is provided, eventually mixing the data formats (byte, half-word, word), writing to the input data register (CRC_INP).

The data stream is generally executed by N concurrent DMA data transfers (mem2mem) where N is less or equal to the number of contexts.

Three standard generator polynomials are given in *Equation 68*, *Equation 69* and *Equation 70* for the CRC computation of each context.

Equation 68CRC-8 VDA CAN)

$$X^8 + X^4 + X^3 + X^2 + 1$$

Equation 69CRC-16-CCITT (x25 protocol)

$$X^{16} + X^{12} + X^5 + 1$$

Equation 70CRC-32 (ethernet protocol)

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

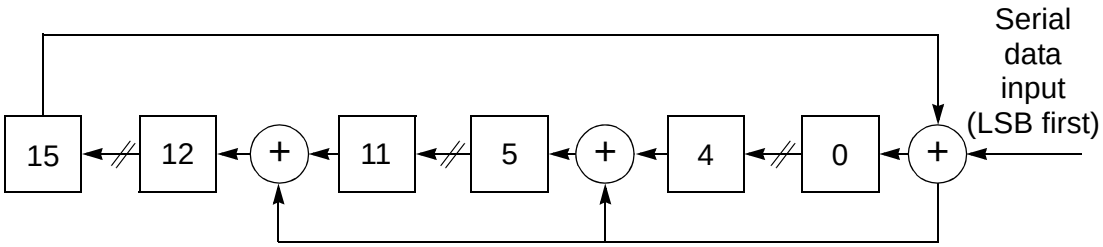


Figure 631. CRC-CCITT engine concept scheme

The initial seed value of the CRC can be programmed initializing the CRC_CSTAT register. The concept scheme (serial data loading) of the CRC engine is given in *Figure 631* for the CRC-CCITT. The design implementation executes the CRC computation in a single clock cycle (parallel data loading). A pipeline scheme has been adopted to de-couple the IPS bus interface from the CRC engine in order to allow the computation of the CRC at speed (zero wait states).

In case of usage of the CRC signature for encapsulation in the data frame of a communication protocol (e.g., SPI, ..) a bit swap (MSB → LSB, LSB → MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register) before to transmit the CRC.

The usage of the CRC is summarized in the flow-chart given in *Figure 632*.

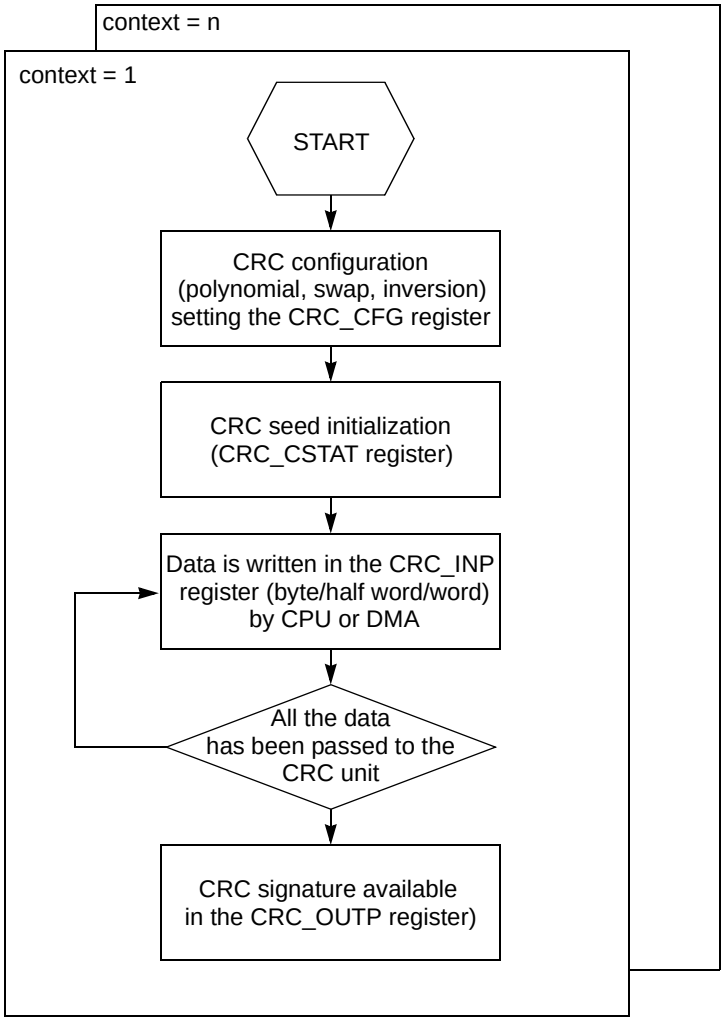


Figure 632. CRC computation flow

34.5 Memory map and registers description

Table 593 shows the CRC memory map.

Table 593. CRC memory map

Offset from CRC_BASE 0xFFE6_8000 (CRC_0) 0x9FE7_0000 (CRC_1)	Register	Location
0x0000	*CRC_CFG—CRC Configuration Register, Context 1*	*on page 1015*
0x0004	*CRC_INP—CRC Input Register, Context 1*	*on page 1016*

Table 593. CRC memory map (continued)

Offset from CRC_BASE 0xFFE6_8000 (CRC_0) 0x9FE7_0000 (CRC_1)	Register	Location
0x0008	*CRC_CSTAT—CRC Current Status Register, Context 1*	*on page 1017*
0x000C	CRC_OUTP—CRC Output Register, *Context 1*	*on page 1018*
0x0010	*CRC_CFG—CRC Configuration Register, Context 2*	*on page 1015*
0x0014	*CRC_INP—CRC Input Register, Context 2*	*on page 1016*
0x0018	*CRC_CSTAT—CRC Current Status Register, Context 2*	*on page 1017*
0x001C	CRC_OUTP—CRC Output Register, *Context 2*	*on page 1018*
0x0020	*CRC_CFG—CRC Configuration Register, Context 3*	*on page 1015*
0x0024	*CRC_INP—CRC Input Register, Context 3*	*on page 1016*
0x0028	*CRC_CSTAT—CRC Current Status Register, Context 3*	*on page 1017*
0x002C	CRC_OUTP—CRC Output Register, *Context 3*	*on page 1018*
0x0030–0x00FF	Reserved	
0x0100	CRC_OUTP_CHK—CRC Output Check Register, Context 1	*on page 1018*
0x0110	CRC_OUTP_CHK—CRC Output Check Register, Context 2	*on page 1018*
0x0120	CRC_OUTP_CHK—CRC Output Check Register, Context 3	*on page 1018*
0x0124–0x3FFF	Reserved	

34.5.1 CRC Configuration Register (CRC_CFG)

Figure 633. CRC Configuration Register (CRC_CFG)

Address: Context 1: Base + 0x0000, Context 2: Base + 0x0010, Context 3: Base + 0x0020

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LEN								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	POLYG		SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 594. CRC_CFG field descriptions

Field	Description
0:7	LEN: *Length of data in payload* Maximum possible data length is 256 (0-255).
8:27	Reserved These are reserved bits. These bits are always read as 0 and must always be written with 0.
28:29	POLYG: *Polynomial selection* 00: CRC-CCITT polynomial 01: CRC-32 polynomial 10: CRC-8 polynomial 11: CRC-8 polynomial This bit can be read andThis bit can be read and written by the software. This bit can be written only during the configuration phase.
30	SWAP: *SWAP selection* 0: No swap selection applied on the CRC_OUTP content 1: Swap selection (MSB -> LSB, LSB -> MSB) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the swap operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.
31	INV: *INV selection* 0: No inversion selection applied on the CRC_OUTP content 1: Inversion selection (bit x bit) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the inversion operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.

34.5.2 CRC Input Register (CRC_INP)

Figure 634. CRC Input Register (CRC_INP)

Address: Context 1: Base + 0x0004
Context 2: Base + 0x0014
Context 3: Base + 0x0024

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 595. CRC_INP field descriptions

Field	Description
0:31	INP: *Input data for the CRC computation* The INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by the software.

34.5.3 CRC Current Status Register (CRC_CSTAT)

Figure 635. CRC Current Status Register (CRC_CSTAT)

Address: Context 1: Base + 0x0008
Context 2: Base + 0x0018
Context 3: Base + 0x0028

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 596. CRC_CSTAT field descriptions

Field	Description
0:31	CSTAT: *Status of the CRC signature* The CSTAT register includes the current status of the CRC signature. No bit swap and inversion are applied to this register. In case of CRC-CCITT polynomial only the16 LSB bits are significant. The 16 MSB bits are tied at 0b during the computation. The CSTAT register can be written at byte, half-word or word. This register can be read and written by the software. This register can be written only during the configuration phase.

34.5.4 CRC Output Register (CRC_OUTP)

Figure 636. CRC Output Register (CRC_OUTP)

Address: Context 1: Base + 0x000C / Context 2: Base + 0x001C / Context 3: Base + 0x002C — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	OUTP															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	OUTP															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 597. CRC_OUTP field descriptions

Field	Description
0:31	OUTP: *Final CRC signature* The OUTP register includes the final signature corresponding to the CRC_CSTAT register value eventually swapped and inverted. In case of CRC-CCITT polynomial only the16 LSB bits are significant. The 16 MSB bits are tied at 0b during the computation. This register can be read by the software.

34.5.5 CRC Output Check Register (CRC_OUTP_CHK)

Figure 637. CRC Output Check Register (CRC_OUTP_CHK)

Address: Context 1: Base + 0x0100 / Context 2: Base + 0x0110 / Context 3: Base + 0x0120 — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	OUTP_CHK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	OUTP_CHK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

34.6 Use cases and limitations

Table 598. CRC_OUTP_CHK field descriptions

Field	Description
0:31	OUTP_CHK: *Output data for the CRC comparison* The OUTP_CHK register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by the software.

Two main use cases shall be considered:

- Calculation of the CRC of the configuration registers during the process safety time
- Calculation of the CRC on the incoming/outgoing frames for the communication protocols (not protected with CRC by definition of the protocol itself) used as a safety-relevant peripheral.

The signature of the configuration registers is computed in a correct way only if these registers do not contain any status bit. Assuming that the DMA engine has N channels (greater or equal to the number of contexts) configurable for the following type of data transfer: mem2mem, periph2mem, mem2periph, the following sequence, as given in *Figure 638*, shall be applied to manage the transmission data flow:

- DMA/CRC module configuration (context x, channel x) by CPU
- Payload transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase1) by DMA (mem2mem data transfer, channel x)
- CRC signature copy from the CRC module (CRC_OUTP register) to the MEM (phase 2) by CPU
- Data block (payload + CRC) transfer from the MEM to the PERIPH module (e.g., SPI Tx fifo) (phase 3) by DMA (mem2periph data transfer, channel x)

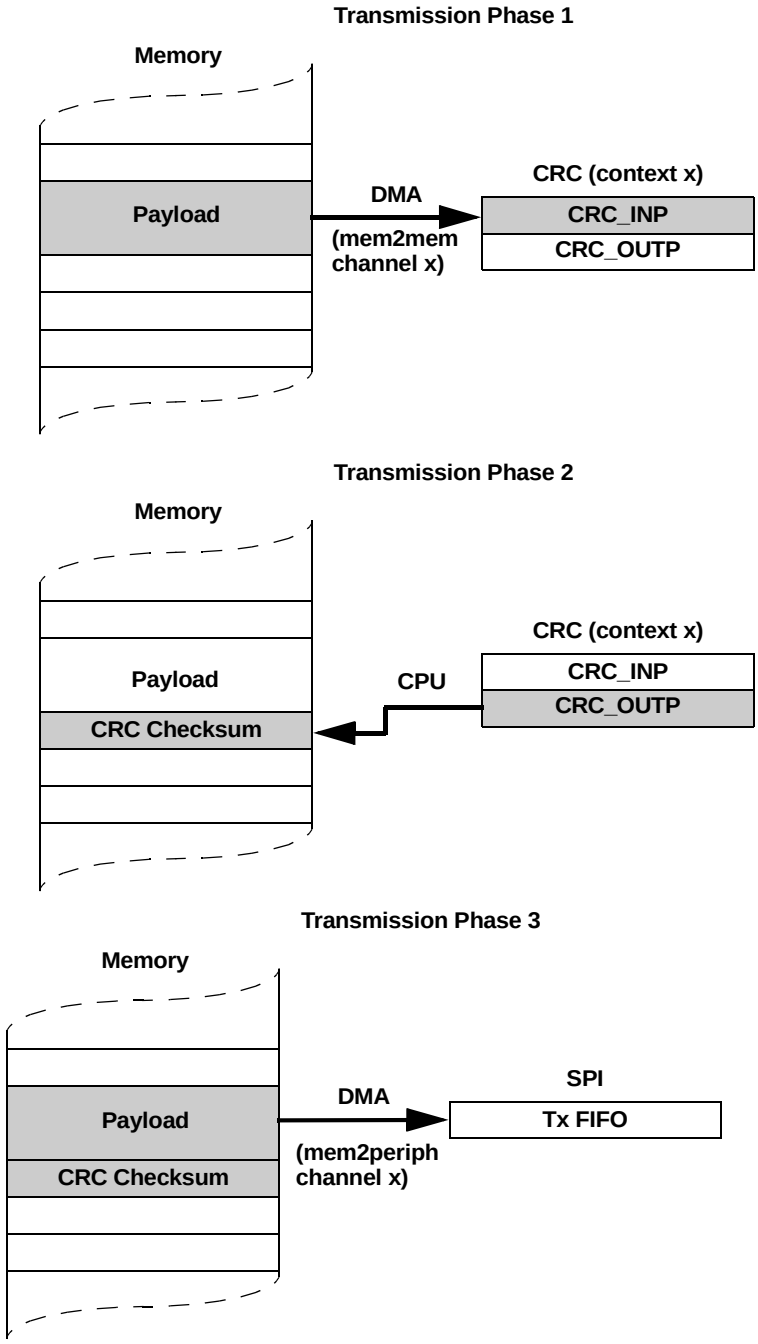


Figure 638. DMA-CRC Transmission Sequence

The following sequence, as given in *Figure 639*, shall be applied to manage the reception data flow:

- DMA/CRC module configuration (context x, channel x) by CPU
- Data block (payload + CRC) transfer from the PERIPH (e.g., SPI Rx fifo) module to the MEM (phase 1) by DMA (periph2mem data transfer, channel x)
- Data block transfer (payload) transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase 2) by DMA (mem2mem data transfer, channel x)
- Data block transfer (CRC) transfer from the MEM to the CRC module (CRC_OUTP_CHK register) to compare with the calculated CRC signature (phase 2) by DMA (mem2mem data transfer, channel x)
- CRC signature check from the CRC module (CRC_OUTP register) by CPU (phase 3)

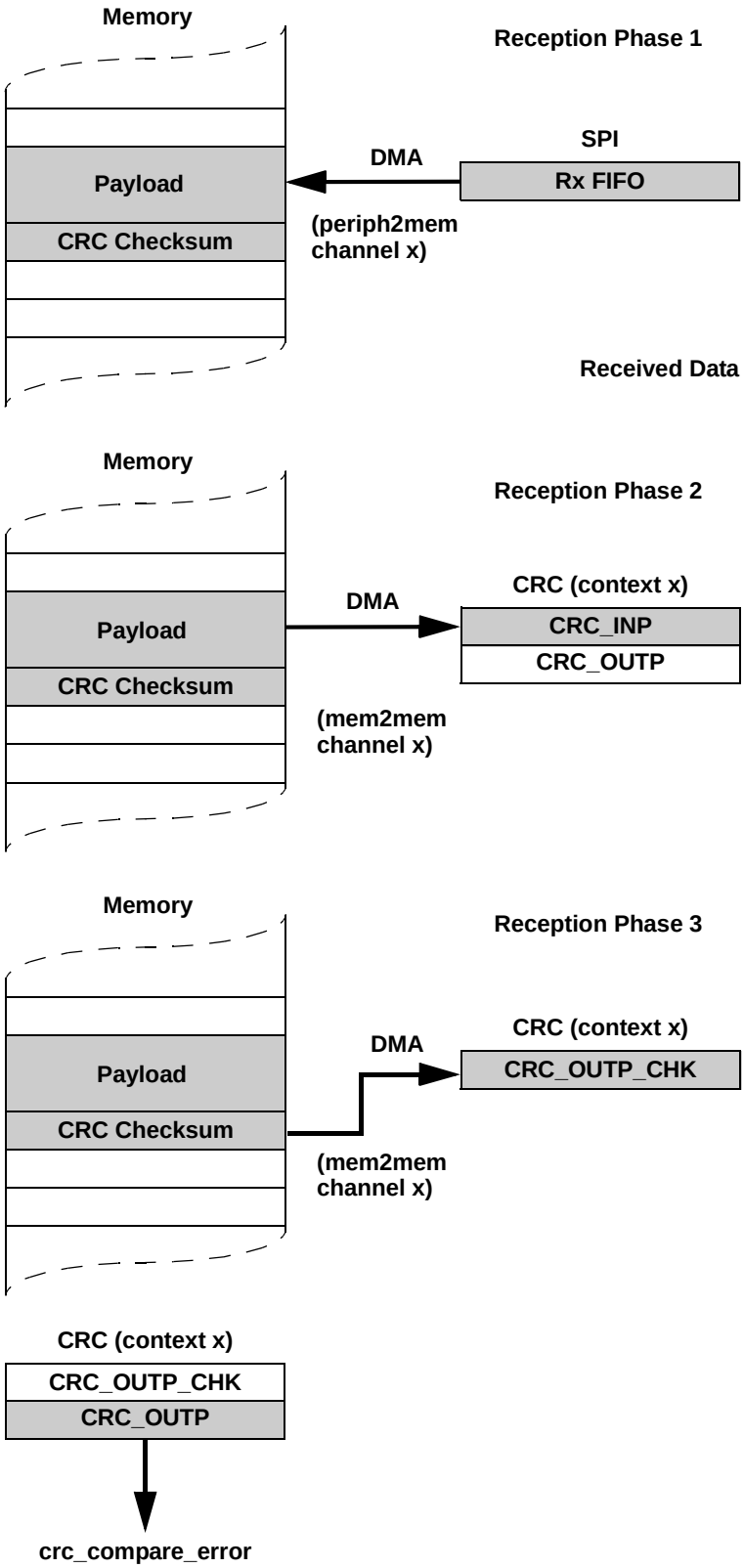


Figure 639. DMA-CRC Reception Sequence

35 Boot Assist Module (BAM)

35.1 Overview

The Boot Assist Module is a block of read-only memory containing VLE code that is executed according to the boot mode of the device.

The BAM allows downloading boot code via the FlexCAN or LINFlex interfaces into internal SRAM and then executing it.

35.2 Features

The BAM provides the following features:

- SPC56xP60/54 in static mode if internal flash is not initialized or invalid
- Programmable 64-bit password protection for serial boot mode
- Serial boot loads the application boot code from a FlexCAN or LINFlex bus into internal SRAM
- Censorship protection for internal flash module

35.3 Boot modes

The SPC56xP60/54 device supports the following boot modes:

- Single Chip (SC) — The device boots from the first bootable section of the Flash main array.
- Serial Boot (SBL) — The device downloads boot code from either LINFlex or FlexCAN interface and then execute it.

If booting is not possible with the selected configuration (e.g., if no Boot ID is found in the selected boot location) then the device enters the static mode.

35.4 Memory map

The BAM code resides in a reserved 8 KB ROM mapped from address 0xFFFF_C000. The address space and memory used by the BAM application is shown in *Table 599*.

Table 599. BAM memory organization

Parameter	Address
BAM entry point	0xFFFF_C000
Downloaded code base address	0x4000_0100

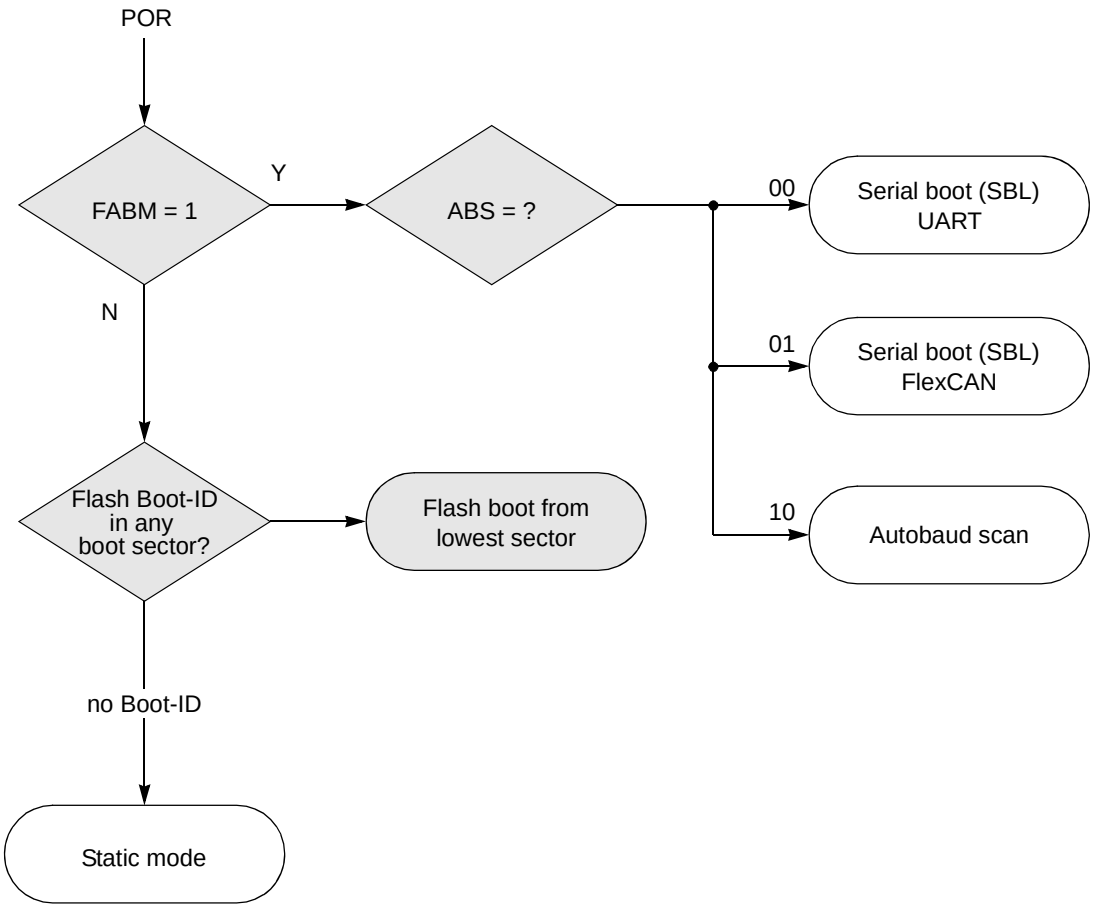
The RAM location where to download the code can be any 4-byte-aligned location starting from the address 0x4000_0100.

35.5 Functional description

35.5.1 Entering boot modes

The SPC56xP60/54 detects the boot mode based on external pins and device status. The following sequence applies (see *Figure 640*):

- To boot either from FlexCAN or LINFlex, the device must be forced into an Alternate Boot Loader Mode via the FAB (Force Alternate Boot Mode), which must be asserted before initiating the reset sequence. The type of alternate boot mode is selected according to the ABS (Alternate Boot Selector) pins (see *Table 600*).
- If FAB is not asserted, the device boots from the lowest Flash sector that contains a valid boot signature.
- If no Flash sector contains a valid boot signature, the device will go into static mode.



Note: The gray blocks in this figure represent hardware actions; white blocks represent software (BAM) actions.

Figure 640. Boot mode selection

Boot configuration pins are:

- PAD A[2] - ABS[0],
- PAD A[3] - ABS[1],
- PAD A[4] - FAB

Table 600. Hardware configuration to select boot mode

FAB	ABS[1:0](1)	Standby-RAM Boot Flag	Boot ID	Boot Mode
1	00	0	—	LINFlex without autobaud
1	01	0	—	FlexCAN without autobaud
1	10	0	—	Scan of both serial interfaces (FlexCAN and LINFlex) with autobaud

1. During reset the boot configuration pins are weak pull down.

35.5.2 SPC56xP60/54 boot pins

The TX/RX pin (LINFlex_0 and FlexCAN_0) used for serial boot and configuration boot pins to select the serial boot mode are described in the *Table 601* for all packages.

Table 601. SPC56xP60/54 boot pins

Port pin	Function	Pin		
		100-pin	144-pin	176-pin(1)
A[2](2)	ABS[0]	57	84	102
A[3](2)	ABS[1]	64	92	116
A[4](2)	FAB	75	108	132
B[0]	CAN_0 TX	76	109	133
B[1]	CAN_0 RX	77	110	134
B[2]	LIN_0 TX	79	114	138
B[3]	LIN_0 RX	80	116	140

1. 176-pin LQFP available only as development package.
2. Weak pull down during reset.

35.5.3 Reset Configuration Half Word (RCHW)

The SPC56xP60/54 Flash is partitioned into boot sectors as shown in *Table 603*.

Each boot sector contains the Reset Configuration Half-Word (RCHW) at offset 0x00.

Figure 641. Reset Configuration Half Word (RCHW)

Address: Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	VLE	BOOT_ID[0:7]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 602. RCHW field descriptions

Field	Description
0-6	Reserved
7 VLE	VLE Indicator This bit configures the MMU for the boot block to execute as either Power Architecture techmology code or as VLE code. 0 Boot code executes as Power Architecture techmology code 1 Boot code executes as VLE code
8-15 BOOT_ID[0:7]	Is valid if its value is 0x5A, then the sector is considered bootable.

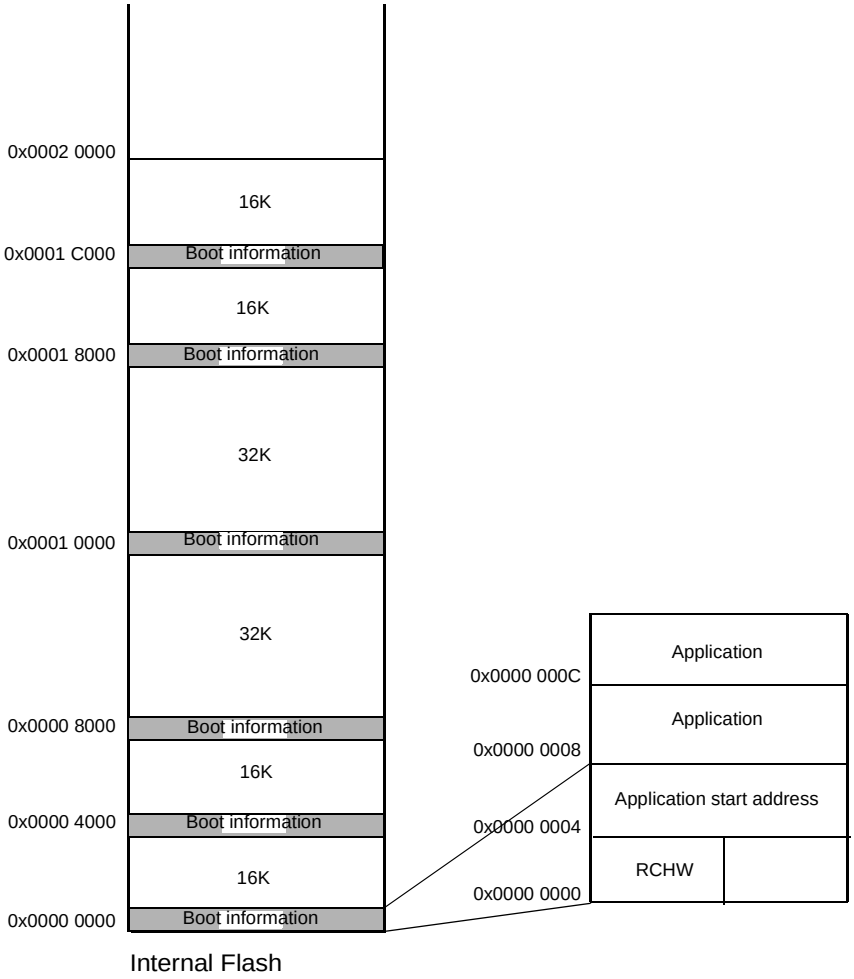


Figure 642. SPC56xP60/54 Flash partitioning and RCHW search

Table 603. Flash boot sector

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0000_8000
3	0x0001_0000
4	0x0001_8000
5	0x0001_C000

35.5.4 Single chip boot mode

In single chip boot mode, the hardware searches the flash boot sector for a valid boot ID. As soon the device detects a bootable sector, it jumps within this sector and reads the 32-bit

word at offset 0x4. This word is the address where the startup code is located (reset boot vector).

Then the device executes this startup code. A user application should have a valid instruction at the reset boot vector address.

If a valid RCHW is not found, the BAM code is executed. In this case BAM moves the SPC56xP60/54 into static mode.

Boot and alternate boot

Some applications require an alternate boot sector in the flash so that the main sector in flash can be erased and reprogrammed in the field.

When an alternate boot is needed, the user can create two bootable sectors. The low sector is the main boot sector and the high sector is the alternate boot sector. The alternate boot sector does not need to be consecutive to the main boot sector.

This ensures that even if one boot sector is erased still there will always be another active boot sector:

- Sector shall be activated (i.e., program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector shall be deactivated writing to 0 some of the bits BOOT_ID bit field (bit1 and/or bit3, and/or bit4, and/or bit6).

35.5.5 Boot through BAM

Executing BAM

Single chip boot mode is managed by hardware and BAM does not participate in it.

BAM is executed only on these two following cases:

- Serial boot mode has been selected by FAB pin
- Hardware has not found a valid Boot-ID in any Flash boot locations

If one of these conditions is true, the device fetches code at location 0xFFFF_C000 and BAM application starts.

BAM software flow

Figure 643 illustrates the BAM logic flow.

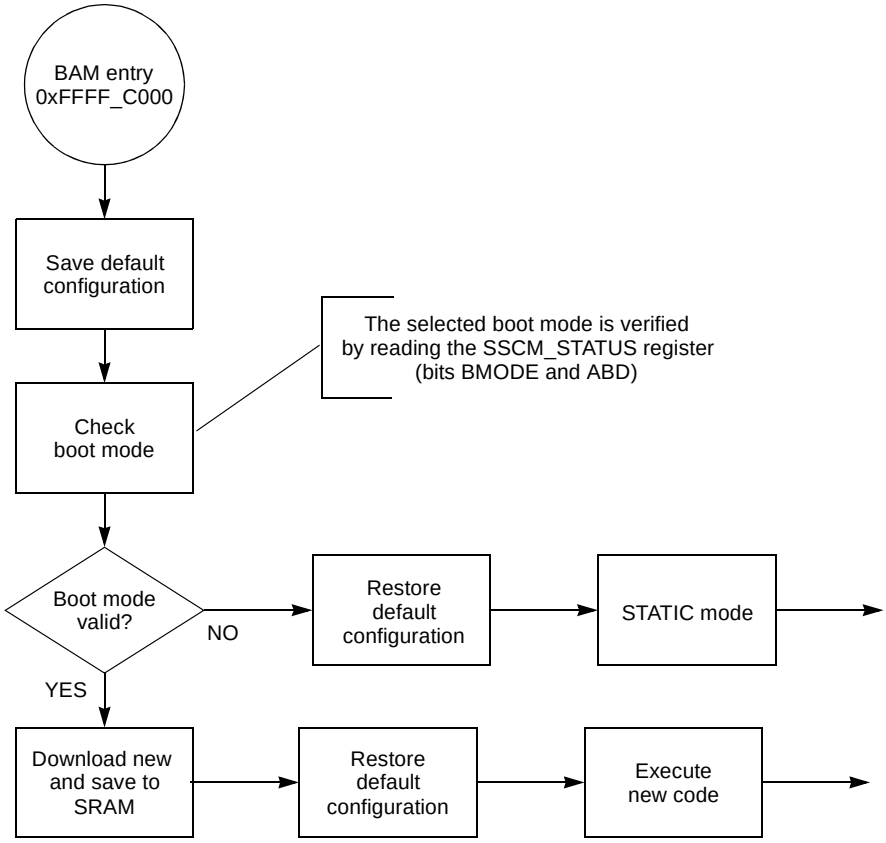


Figure 643. BAM logic flow

The first action is to save the initial device configuration. In this way is possible to restore the initial configuration after downloading the new code but before executing it. This allows the new code being executed as the device was just coming out of reset.

The BMODE and ABD fields of SSCM_STATUS register (see *Section : System Status register (STATUS)*) indicate which boot has to be executed (see *Table 604*).

If the BMODE field shows either a single chip value (011) or a reserved value, the boot mode is not considered valid and the BAM pushes the device into static mode.

In all other cases the code of the relative boot is called. Data is downloaded and saved into proper SRAM location.

Table 604. Fields of SSCM STATUS register used by BAM

Field	Description
BMODE [2:0]	Device Boot Mode 000 Test Flash/autobaud_scan 001 CAN Serial Boot Loader 010 SCI Serial Boot Loader 011 Single Chip 100–111Reserved This field is updated only during reset.

Then, the initial device configuration is restored and the code jumps to the address of downloaded code. At this point BAM has just finished its task.

If an error occurs, (e.g., communication error, wrong boot selected, etc.), the BAM restores the default configuration and puts the device into static mode. Static mode means the device enters the low power mode SAFE and the processor executes a wait instruction. This is needed if the device cannot boot in the selected mode. During BAM execution and after, the mode reported by the field S_CURRENT_MODE of the register ME_GS in the module ME Module is “DRUN”.

BAM resources

BAM uses/initializes the following MCU resources:

- ME and CGM modules to initialize mode and clock sources
- CAN_0, LINFlex_0, and their pads when performing serial boot mode
- SSCM to check the boot mode and during password check (see *Table 604* and *Figure 644*)
- External oscillator

The following hardware resources are used only when autobaud feature is selected:

- STM to measure the baud rate
- CMU to measure the external clock frequency related to the internal RC clock source
- FMPLL to work with system clock near the maximum allowed frequency (this to have higher resolution during baud rate measurement).

As already mentioned, the initial configuration is restored before executing the downloaded code.

When the autobaud feature is disabled, the system clock is selected directly from the external oscillator. Thus the oscillator frequency defines baud rates for serial interfaces used to download the user application (see *Table 605*).

Table 605. Serial boot mode without autobaud—baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/s)
f_{extal}	f_{extal} / 833	f_{extal} / 40
8	9600	200 K
12	14400	300 K

Table 605. Serial boot mode without autobaud—baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/s)
16	19200	400 K
20	24000	500 K
40	48000	1 M

Download and execute the new code

From a high level perspective, the download protocol follows these steps:

1. Send message and receive acknowledge message for autobaud or autobit rate selection. (optional step).
2. Send 64-bit password.
3. Send start address, size of downloaded code in bytes, and VLE bit[m].
4. Download data.
5. Execute code from start address.

Each step must be complete before the next step starts.

The step from 2 to 5 are correct if autobaud is disabled. Otherwise, to measure the baud rate, some data is sent from the host to the MCU before step 2 (see *Section 35.6.1: Autobaud feature*).

The communication is done in half duplex manner. Any transmission from the host is followed by the MCU transmission:

1. Host sends data to MCU and start waiting.
2. MCU echoes to host the data received.
3. MCU verifies if echo is correct.
   - If data is correct, the host can continue to send data.
   - If data is not correct, the host stops transmitting and the MCU needs to be reset.

All multi-byte data structures are sent MSB first.

A more detailed description of these steps follows.

Download 64-bit password and password check

The first 64 received bits represent the password. This password is sent to the Password Check procedure for verification.

Password check data flow is shown in *Figure 644* where:

- SSCM_STATUS[SEC] = 1 means flash secured
- SSCM_STATUS[PUB] = 1 means flash with public access.

In case of flash with public access, the received password is compared with the public password 0xFEED_FACE_CAFE_BEEF.

m. Since the device supports only VLE code and does not support Book E code, this flag is used only for backward compatibility.

If public access is not allowed but the flash is not secured, the received password is compared with the value saved on NVPWD0 and NVPWD1 registers.

In uncensored devices, it is possible to download code via LINFlex or FlexCAN (serial boot mode) into internal SRAM with any 64-bit private password stored in the flash and provided during the boot sequence.

In the previous cases, comparison is done by the BAM application. If it goes wrong, BAM pushes the device into static mode.

In case of public access not allowed and flash secured, the password is written into SSCM[PWCMPH/L] registers.

In case of flash secured with public access not allowed (user password configured in the NVPWD0 and NVPWD1 registers), the user must set the swapped password: NVPWD1 and NVPWD0 to access the device (refer to following examples).

Example 16

In devices with flash secured, registers are programmed:

NVPWD0 = 0x87654321

NVPWD1 = 0x12345678

NVSCI0 = 0x55AA1111

NVSCI1 = 0x55AA1111

To download the code via SLB, the provided password is 0x1234_5678_8765_4321 (swapped WITH respect to NVPWD0/NVPWD1 —> NVPWD1/NVPWD0).

Example 17

In devices with flash NOT secured, registers are programmed:

NVPWD0 = 0x87654321

NVPWD1 = 0x12345678

NVSCI0 = 0x55AA55AA

NVSCI1 = 0x55AA55AA

To download the code via SLB the provided password is 0x8765_4321_1234_5678 (as expected from NVPWD0/NVPWD1).

After a fixed time waiting, comparison is done by hardware. Then BAM again verifies the SEC flag in SSCM_STATUS:

- SEC = 0, flash is now unsecured and BAM continues its task
- SEC = 1, flash is still secured because password was wrong; BAM puts the device into static mode.

This fixed time depends on external crystal oscillator frequency (XOSC). With XOSC of 12 MHz, fixed time is 350 ms.

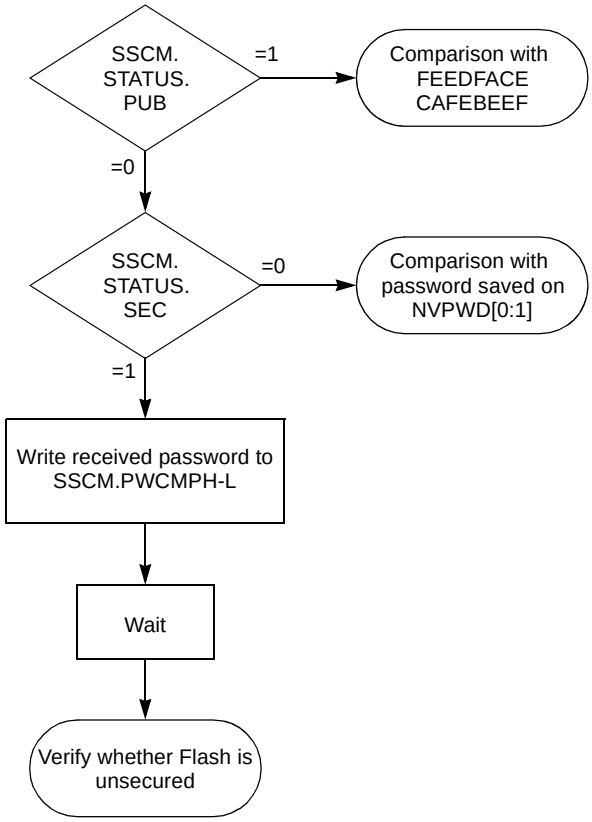


Figure 644. Password check flow

Download start address, VLE bit and code size

The next 8 bytes received by the MCU contain a 32-bit Start Address, the VLE mode bit and a 31-bit code Length as shown in *Figure 645*.

The VLE bit (Variable Length Instruction) indicates which instruction set for which the code has been compiled. This device family supports only VLE = 1. The bit is used for backward compatibility.

The Start Address defines where the received data will be stored and where the MCU will branch after the download is finished. The two LSB bits of the Start Address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

The Length defines how many data bytes have to be loaded.

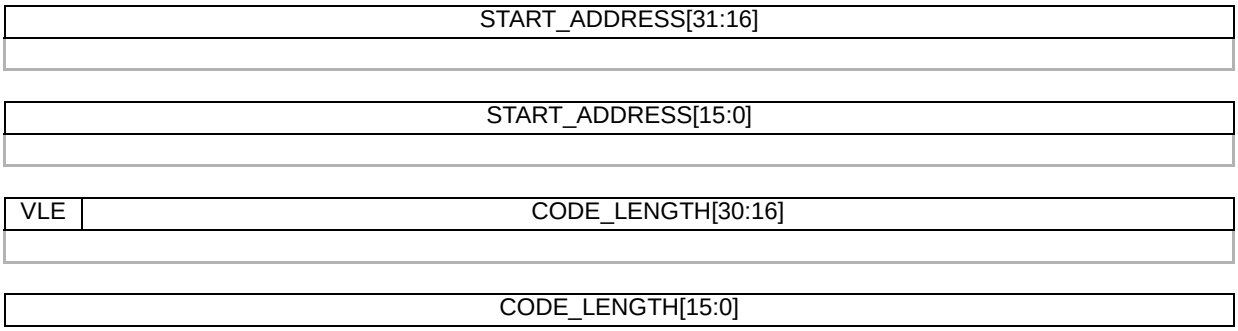


Figure 645. Start address, VLE bit and download size in bytes

Download data

Each byte of data received is stored into device's SRAM, starting from the address specified in the previous protocol step.

The address increments until the number of bytes of data received matches the number of bytes specified in the previous protocol step.

Since the SRAM is protected by 32-bit wide Error Correction Code (ECC), BAM always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, BAM fills it with 0 bytes.

Then a "dummy" word (0x0000_0000) is written to avoid ECC error during core prefetch.

Execute code

The BAM program waits for the last echo message transmission being completed.

Then it restores the initial MCU configuration and jumps to the loaded code at Start Address that was received in step 3 of the protocol.

At this point BAM has finished its tasks and MCU is controlled by new code executing from SRAM.

35.5.6 Boot from UART—autobaud disabled

Configuration

Boot from UART protocol is implemented by LINFlex_0 module. Pins used are:

- LINFlex_TX corresponds to pin B[2]
- LINFlex_RX corresponds to pin B[3].

When autobaud feature is disabled, the system clock is driven by external oscillator.

LINFlex controller is configured to operate at a baud rate = system clock frequency/833 (see *Table 605* for baud rate example), using 8-bit data frame without parity bit and 1 stop bit.

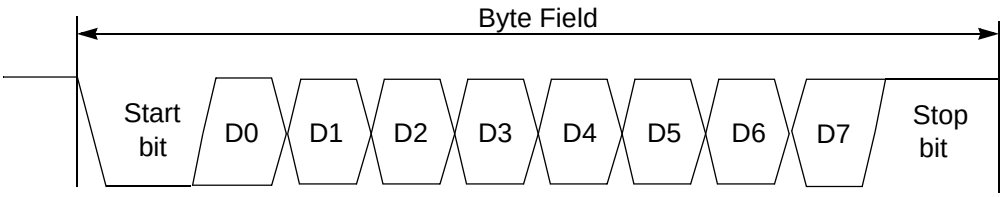


Figure 646. LINFlex bit timing in UART mode

UART boot mode download protocol

Table 606 summarizes the download protocol and BAM action during the UART boot mode.

Table 606. UART boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	64-bit password (MSB first)	64-bit password	Password checked for validity and compared against stored password.
2	32-bit store address	32-bit store address	Load address is stored for future use.
3	VLE bit + 31-bit number of bytes (MSB first)	VLE bit + 31-bit number of bytes (MSB first)	Size of download is stored for future use. Verify if VLE bit is set to 1.
4	8 bits of raw binary data	8 bits of raw binary data	8-bit data are packed into 32-bit word. This word is saved into SRAM starting from the “Load address”. “Load address” increments until the number of data received and stored matches the size as specified in the previous step.
5	none	none	Branch to downloaded code.

35.5.7 Bootstrap with FlexCAN—autobaud disabled

Configuration

Boot from FlexCAN protocol is implemented by the FlexCAN_0 module. Pins used are:

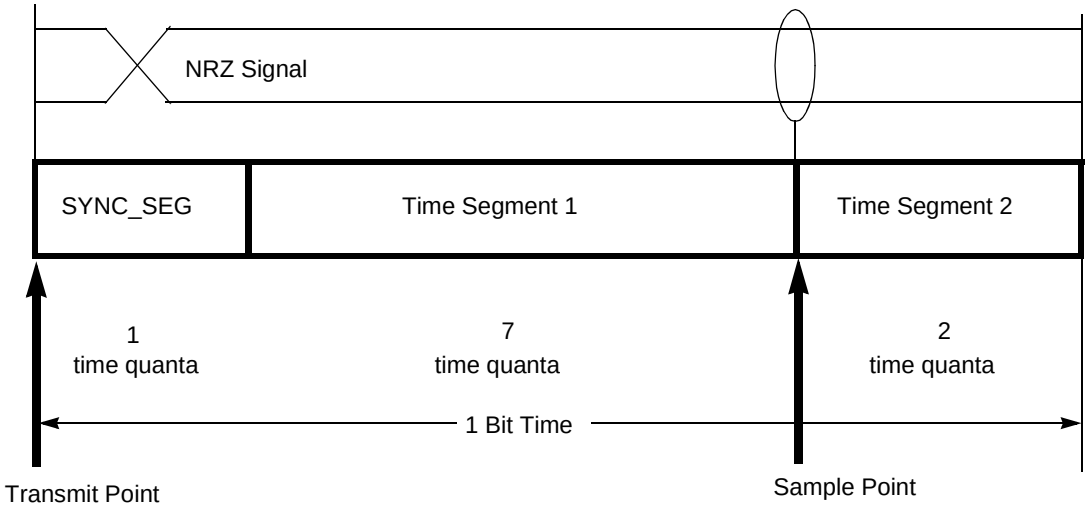
- CAN_TX corresponds to pin B[0]
- CAN_RX corresponds to pin B[1].

Boot from FlexCAN with autobaud disabled uses system clock driven by the external oscillator.

The FlexCAN controller is configured to operate at a baud rate equal to the system clock frequency/40 (see *Table 605* for examples of baud rate).

It uses the standard 11-bit identifier format detailed in the FlexCAN 2.0A specification.

FlexCAN controller bit timing is programmed with 10 time quanta, and the sample point is 2 time quanta before the end, as shown in *Figure 647*.



1 time Quanta = 4 system clock periods

Figure 647. FlexCAN bit timing

35.6 FlexCAN boot mode download protocol

Table 607 summarizes the download protocol and BAM action during the FlexCAN boot mode. All data are transmitted bytewise.

Table 607. FlexCAN boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	FlexCAN ID 0x011 + 64-bit password	FlexCAN ID 0x001 + 64-bit password	Password checked for validity and compared against stored password.
2	FlexCAN ID 0x012 + 32-bit store address + VLE bit+ 31-bit number of bytes	FlexCAN ID 0x002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address is stored for future use. Size of download is stored for future use. Verify if VLE bit is set to 1.
3	FlexCAN ID 0x013 + 8 to 64 bits of raw binary data	FlexCAN ID 0x003 + 8 to 64 bits of raw binary data	8-bit data are packed into 32-bit words. These words are saved into SRAM starting from the “Load address”. “Load address” increments until the number of data received and stored matches the size as specified in the previous step.
4	none	none	Branch to downloaded code.

35.6.1 Autobaud feature

The autobaud feature allows boot operation with a wide range of baud rates independent of the external oscillator frequency.

Configuration

SPC56xP60/54 devices implement the autobaud feature via FlexCAN or LINFlex selecting the active serial communication peripheral by means of an autoscan routine.

When autobaud configuration is selected by ABS and FAB pins, the autoscan routine starts and listens to the active bus protocol. Initially the LinFlex_0 RX pin and FlexCAN_0 RX pin are configured as GPIO inputs:

- for 176-pin, 144-pin, and 100-pin LQFP packages internal weak pull-up enabled for both RX pins

The autoscan routine waits in polling for the first LOW level to select which routine will be executed:

- FlexCAN Autobaud routine
- LinFlex Autobaud routine

Then the measurement baud rate is computed to configure the serial communication at the right rate. In the end of baud rate measurement, LinFlex_0 RX pin and FlexCAN_0 RX pin switches to work as dedicated pin.

Baud rate measurement is using the System Timer Module (STM) which is driven by the system clock. Measurement itself is performed by software polling the related inputs as general purpose IO's, resulting in a detection granularity that is directly related to the execution speed of the software.

One main difference of the autobaud feature is that the system clock is not driven directly by the external oscillator, but it is driven by the FMPLL output. The reason is that to have an optimum resolution for baud rate measurement, the system clock needs to be nearer to the maximum allowed device's frequency.

This is achieved with the following two steps:

1. using the Clock Monitor Unit (CMU) and the internal RC oscillator (IRC), the external frequency is measured using the IRC as reference to determine this frequency.
2. Based on the result of this measurement, the FMPLL is programmed to generate a system clock that is configured to be near, but lower, to the maximum allowed frequency.

The relation between system clock frequency and external clock frequency with FMPLL configuration value is shown in *Table 608*.

Table 608. System clock frequency related to external clock frequency

f_{osc} [MHz]	f_{rc}/f_{osc} (1)	f_{sys} [MHz]
4–8	4–2	16–32
8–12	2–4/3	32–48
12–16	4/3–1	36–48
16–24	1–2/3	32–48
> 24	< 2/3	> 24

1. These values and consequently the f_{sys} suffer from the precision of RC internal oscillator used to measure f_{osc} through CMU module.

After setting up the system clock, the BAM autoscan code configures the FlexCAN RX pin (B[1] on all packages) and LINFlex RX pin (B[3] on LQFP100 or B[7] on LQFP64) as GPIO inputs and searches for FlexCAN RX pin level to verify if CAN is connected or not.

Then continuously waits in polling on change of RX pins level.The FlexCAN RX pin level takes precedence. First signal found at low level selects the serial boot routine that will be executed.

In case a low level is detected on any input, the corresponding autobaud measurement functionality is started:

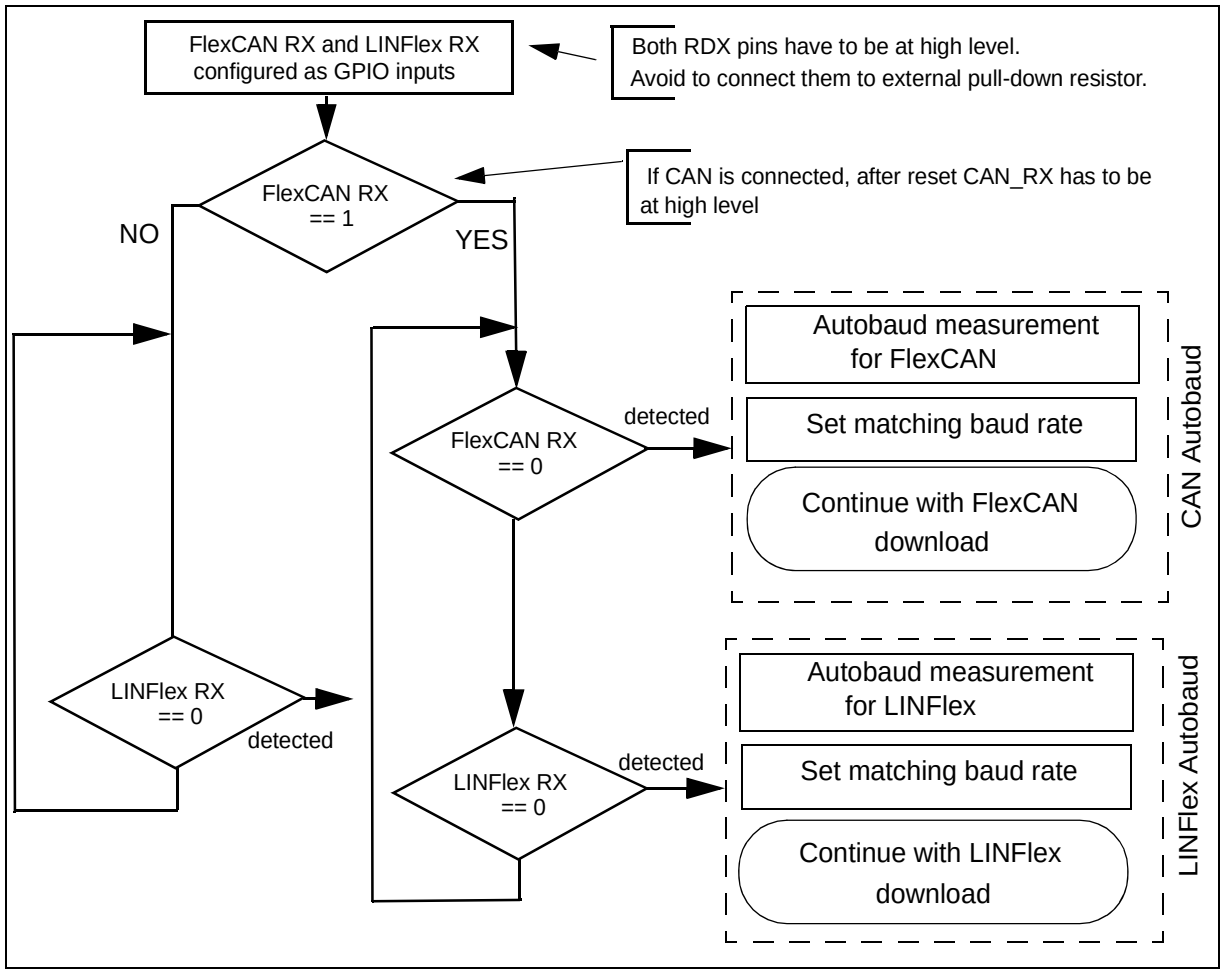
- when FlexCAN RX (corresponds to pin B[1]) level is low, the CAN autobaud measurement starts and then sets up the FlexCAN baud rate accordingly;
- when UART RX (corresponds to pin B[3] on LQFP100 or B[7] on LQFP64) level is low, the UART autobaud measurement starts and then sets up the LINFlex baud rate accordingly.

After performing the autobaud measurement and setting up the baud rate, the corresponding RX input is reconfigured and the related standard download process is started; in case of a detected CAN transmission a download using the CAN protocol as described in section *Section 35.5.7: Bootstrap with FlexCAN—autobaud disabled*, and in case of a detected UART transmission a download using the UART protocol as described in *Section 35.5.6: Boot from UART—autobaud disabled*.

The following *Figure 648* identifies the corresponding flow and steps.

Note: When autobaud scan is selected, initially both LINFlex_0 RX pin and FlexCAN_0 RX pin should be at high level. No external circuity should pull-down them to allow right autoscan.

Figure 648. BAM Autoscan code flow



Boot from UART with autobaud enabled

The only difference between booting from UART with autobaud enabled and booting from UART with autobaud disabled is that a further byte is sent from the host to the MCU when autobaud is enabled. The value of that byte is 0x00.

This first byte measures the time from falling edge and rising edge. The baud rate can be calculated from this time.

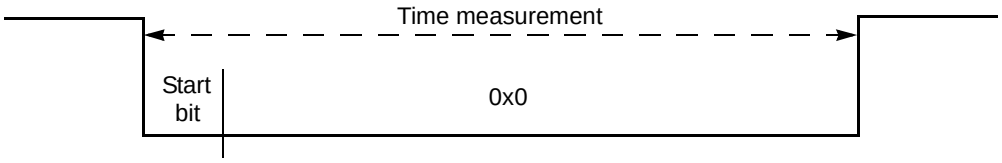


Figure 649. Baud measurement on UART boot

Initially the UART RX pin is configured as GPIO input and it waits in polling for the first falling edge, then STM starts. UART RX pin waits again for the first rising. Then STM stops and from its measurement baud rate is computed.

The LINFlex module is configured to work in UART mode with the calculated baud rate. Then an acknowledge byte (0x59, ASCII char “Y”) is sent.

From this point, the BAM follows the normal UART mode boot protocol (see *Figure 650*).

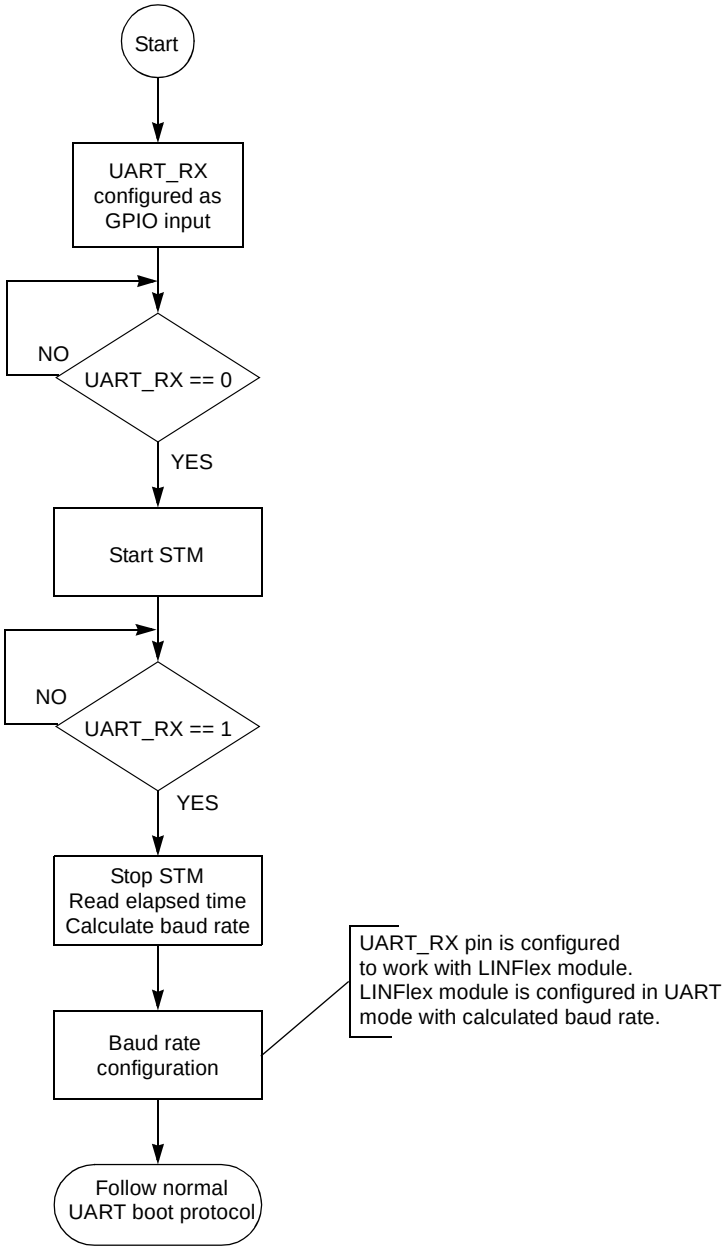


Figure 650. BAM rate measurement flow during UART boot

Choosing the host baud rate

The calculation of the UART baud rate from the length of the first 0 byte that is received, allows the operation of the boot loader with a wide range of baud rates. However, to ensure proper data transfer, the upper and lower limits have to be kept.

SCI autobaud rate feature operates by polling the LINFlex_RX pin for a low signal. The length of time until the next low to high transition is measured using the System Timer Module (STM) time base. This high-low-high transition is expected to be a zero byte: a start bit (low) followed by eight data bits (low) followed by a stop bit (high).

Upon reception of a zero byte and configuration of the baud rate, an acknowledge byte is returned to the host using the selected baud rate.

Time base is enabled at reception of first low bit, disabled and read at reception of next high bit. Error introduced due to polling will be small (typically < 6 cycles).

The following equation gives the relation between baud rate and LINFlex register configuration:

Equation 71

$$LDIV = \frac{F_{cpu}}{16 \cdot baudrate}$$

LDIV is an unsigned fixed point number and its mantissa is coded into 13 bits of the LINFlex's register LINIBRR.

From this equation and considering that a single UART transmission contains 9 bits, it is possible to obtain the connection between time base measured by STM and LINIBB register:

Equation 72

$$LINIBRR = \frac{timebase}{144}$$

To minimize errors in baud rate, a large external oscillator frequency value and low baud rate signal are preferred.

Example 18 Baud rate calculation for 24 kBaud signal

Considering a 24 kbaud signal and the device operating with 20 MHz external frequency.

Over 9 bits the STM will measure: (9 × 20 MHz)/24 kbaud = 7497 cycles.

Error expected to be approximately ± 6 cycles due to polling.

Thus, LINIBB will be set to 52 (rounding required). This results in a baud rate of 24.038 kbaud. Error of < 0.2%.

To maintain the maximum deviation between host and calculated baud rate, recommendations for the user are listed *Table 609*.

Table 609. Maximum and minimum recommended baud rates

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for guaranteed < 2.5% deviation	Min baud rate for guaranteed < 2.5% deviation
4	13.4 Kbit/s (SBR = 19)	30 bit/s (SBR = 8192)
8	26.9 Kbit/s (SBR = 19)	60 bit/s (SBR = 8192)
12	38.4 Kbit/s (SBR = 20)	90 bit/s (SBR = 8192)

Table 609. Maximum and minimum recommended baud rates (continued)

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for guaranteed < 2.5% deviation	Min baud rate for guaranteed < 2.5% deviation
16	51.2 Kbit/s (SBR = 20)	120 bit/s (SBR = 8192)
20	64.0 Kbit/s (SBR = 20)	150 bit/s (SBR = 8192)

Higher baud rates high may be used, but the user will be required to ensure they fall within an acceptable error range. This is illustrated in *Figure 651*, which shows the effect of quantization error on the baud rate selection.

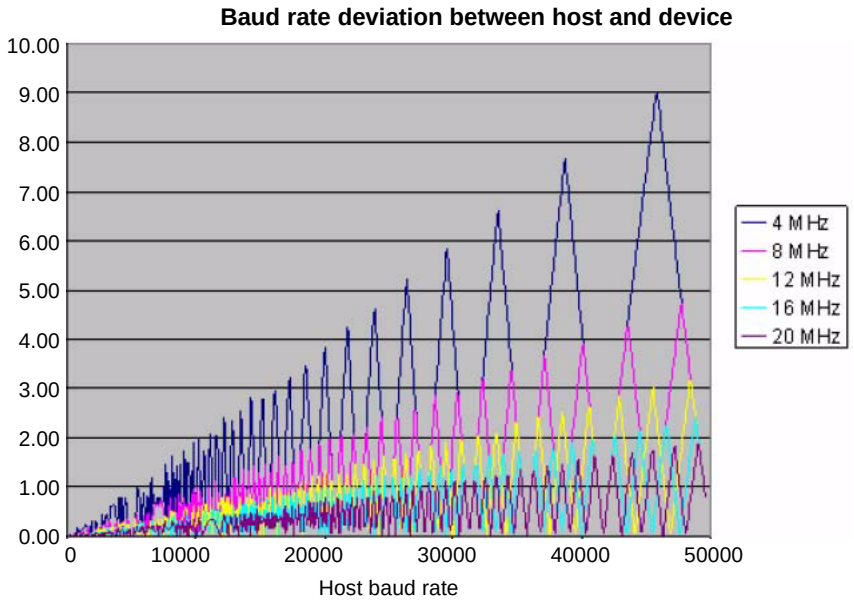


Figure 651. Baud rate deviation between host and SPC56xP60/54

Example 19 Baud rate calculation for 250 kBaud signal

Considering reception of a 250kBaud signal from the host and SPC56xP60/54 operating with a 4 MHz oscillator. Over 9 bits the time base will measure: (9 × 4e6)/250e3 = 144 cycles.

Thus, LINIBB is set to 144/144 = 1. This results in a baud rate of exactly 250 kBd.

However, a slower 225 kBd signal operating with 4 MHz XTAL would again result in LINIBB = 1, but this time with an 11.1% deviation.

Boot from FlexCAN with autobaud enabled

The only difference between booting from FlexCAN with autobaud enabled and booting from FlexCAN with autobaud disabled is that the following initialization FlexCAN frame is sent for baud measurement purposes from the host to the MCU when autobaud is enabled:

- Standard identifier = 0x0,
- Data Length Code (DLC) = 0x0.

As all the bits to be transmitted are dominant bits, there is a succession of 5 dominant bits and 1 stuff bit on the FlexCAN network (see *Figure 652*).

From the duration of this frame, the MCU calculates the corresponding baud rate factor with respect to the current CPU clock and initializes the FlexCAN interface accordingly.

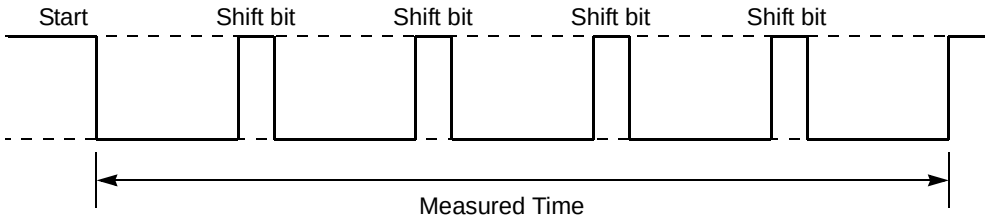


Figure 652. Bit time measure

In FlexCAN boot mode, the FlexCAN RX pin is first configured to work as a GPIO input. In the end of baud rate measurement, it switches to work with the FlexCAN module.

The baud rate measurement flow is detailed in *Figure 653*.

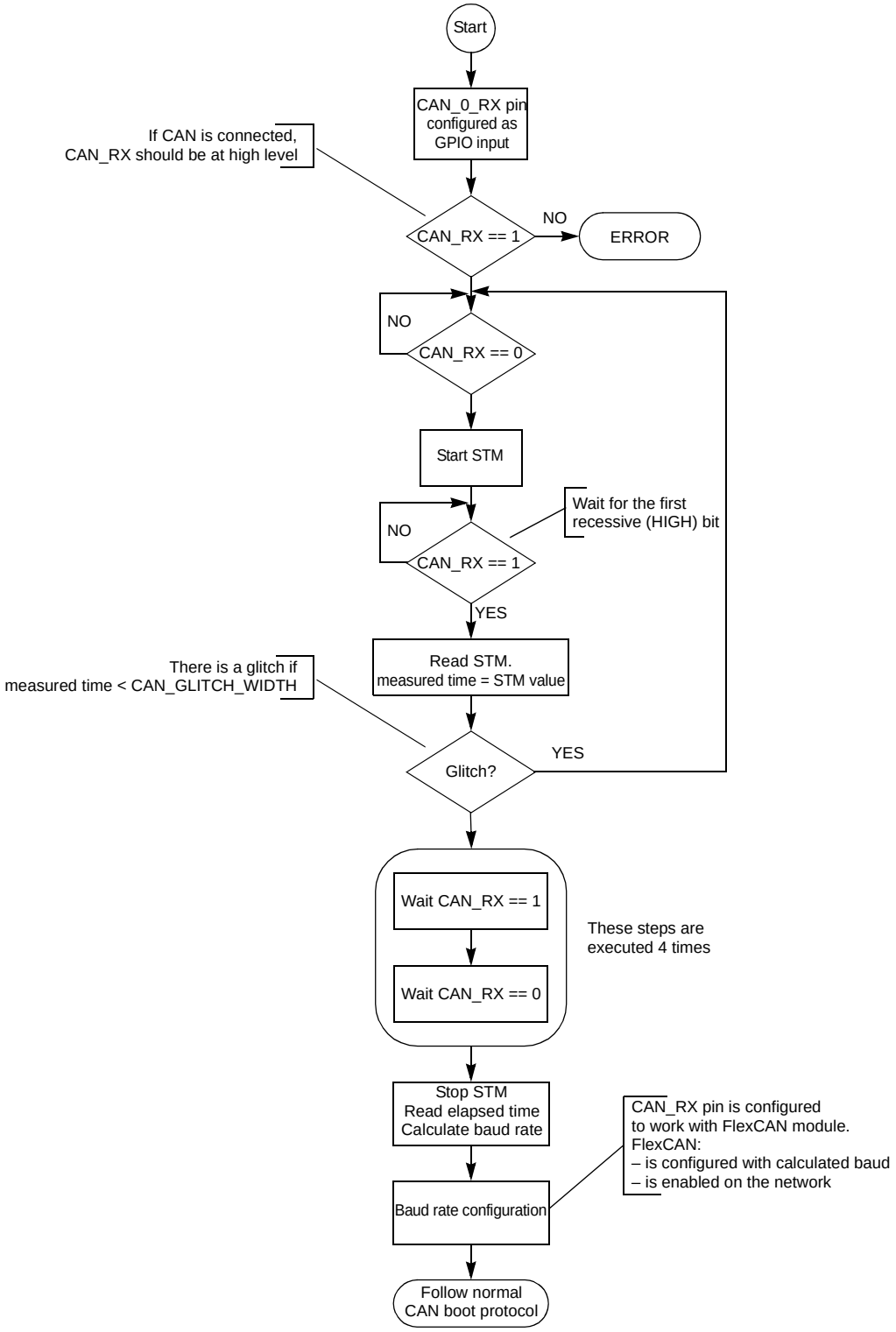


Figure 653. BAM rate measurement flow during FlexCAN boot

Choosing the host baud rate

The calculation of the FlexCAN baud rate allows the operation of the boot loader with a wide range of baud rates. However, to ensure proper data transfer, the upper and lower limits have to be kept.

Pins are measured until reception of the 5th recessive bit. Thus a total of 29 bit-times are measured (see *Figure 652*).

When calculating bit times and prescalers, to minimize any errors, ideally operate with the minimum system clock prescaler divider (CAN_CR[PRESDIV]) and maximum number of time quanta possible.

After measuring the 29 bit times, the results stored in the STM time base are used to select PRESDIV. The number of time quanta in a FlexCAN bit time is given by:

Bit_time = SYNCSEG + TSEG1 + TESG2

SYNCSEG = Exactly one time quantum.

TSEG1 = PROGPSEG + PSEG1 + 2

TSEG2 = PSEG2 + 1

Time base result = 29 × (Presdiv+1) × (SYNCSEG + TSEG1 + TSEG2)

FlexCAN protocol specifies that the FlexCAN bit timing should comprise a minimum of 8 time quanta and a maximum of 25 time quanta. Therefore, the available range is:

$8 \leq 1 + TSEG1 + TESG2 \leq 25$

For 29 bit times, the possible range in which the result in the time base may lie, accounting for PRESDIV, is:

$(232 \times (1 + PRESDIV)) \leq time\ base \leq (725 \times (1 + PRESDIV))$

Therefore, the available values of the time base can be divided into windows of 725 counts.

Table 610. Prescaler/divider and time base values

PRESDIV	Time base Minimum	Time base Maximum
0	232	725
1	726	1450
2	1451	2175
3	2176	2900

In the BAM, the time base is divided by 726, the remainder is discarded. The result provides the CAN_CR[PRESDIV] to be selected.

To help compensate for any error in the calculated baud rate, the resynchronization jump width will be increased from its default value of 1 to a fixed value of 2 time quanta.This is the maximum value allowed that can accommodate all permissible can baud rates. See *Table 611*.

Table 611. FlexCAN standard compliant bit timing segment settings

Time Segment 1	Time Segment 2	RJW
5..10	2	1..2
4..11	3	1..3
5..12	4	1..4
6..13	5	1..4
7..14	6	1..4
8..15	7	1..4
9..16	8	1..4

Timing segment 2 is kept as large as possible to keep sample time within bit time.

Table 612. Lookup table for FlexCAN bit timings

Desired number of Time quanta (DTq)	Time Segment 2	Time segment 1	
	PSEG2+1	PSEG1+1	PROPSEG+1
8 to 13(1)	2	2	DTq-5
8 to 13(2)	3	2	DTq-6
14 to 15	3	3	DTq-6
16 to 17	4	4	DTq-7
18 to 19	5	5	DTq-8
20 to 21	6	6	DTq-9
22 to 23	7	7	DTq-10
24 to 25	8	8	DTq-11

1. PRESDIV+1 > 1
2. PRESDIV+1 = 1 (to accommodate information processing time IPT of 3 tq) Note: All TSEG1 and TSEG2 times have been chosen to preserve a sample time between 70% and 85% of the bit time.

Table 613. PRESDIV + 1 = 1

Desired number of time quanta	Register contents for CANA_CR
8	0x004A_2001
9	0x004A_2002
10	0x004A_2003
11	0x004A_2004
12	0x004A_2005
13	0x004A_2006

Table 614. PRESDIV + 1 > 1 (YY = PRESDIV)

Desired number of time quanta	Register contents for CANA_CR
8	0xYY49_2002
9	0xYY49_2003
10	0xYY49_2004
11	0xYY49_2005
12	0xYY49_2006
13	0xYY49_2007
14	0xYY52_2007
15	0xYY52_2008
16	0xYY5B_2008
17	0xYY5B_2009
18	0xYY64_2009
19	0xYY64_200A
20	0xYY6D_200A
21	0xYY6D_200B
22	0xYY76_200B
23	0xYY76_200C
24	0xYY7F_200C
25	0xYY7F_200D

Worked examples showing FlexCAN Autobaud rate:

Example 208 MHz crystal

Consider case where using an 8 MHz crystal, user attempts to send 1 MB (max permissible baud rate) FlexCAN message.

– Time base, clocking at crystal frequency, would measure:
– 1MB = 8 clocks/bit => 29 * 8 = 232 clocks
– To calculate PRESDIV = 232/725 =>PRESDIV = 0
– To calculate time quanta requirement:
– Time base result = 29 *(Presdiv+1) * (SYNCSEG + TSEG1 + TSEG2)
– 232 = 29 * 1 * (1 + TSEG1 + TSEG2)
– 1 + TSEG1 + TSEG2 = 8.
– From the lookup table, CANA_CR = 0x004A_2001.
– This give a baud rate of X. This give 0% error.

Example 2120 MHz crystal

Consider case where using a 20 MHz crystal, user attempts to send 62.5 Kb/s FlexCAN message.

- Time base, clocking at crystal frequency, would measure:
- 62.5 Kb/s = 320 clocks/bit => 29 * 320 = 9280 clocks
- To calculate PRESDIV = 9280/725 = 12r580 => PRESDIV = 12
- To calculate time quanta requirement:
- Time base result = 29 x (Presdiv+1) x (SYNCSEG + TSEG1 + TSEG2)
- 9280 = 29 x 13 x (1 + TSEG1 + TSEG2)
- 1 + TSEG1 + TSEG2 = 24.6~ 25 (need to round up - S/W must track remainder)
- From the lookup table, CANA_CR = 0x0C7F_200D.
- This give a baud rate of 61.538k Baud
- This equates to an error of: ~1.6%

This excludes cycle count error introduced due to software polling likely to be ~6 system clocks.

35.6.2 Interrupt

No interrupts are generated by or are enabled by the BAM.

35.7 Censorship

Censorship can be enabled to protect the contents of the flash memory from being read or modified. In order to achieve this, the censorship mechanism controls access to the:

- JTAG / Nexus debug interface
- Serial boot mode (which could otherwise be used to download and execute code to query or modify the flash memory)

To re-gain access to the flash memory via JTAG or serial boot, a 64-bit password must be correctly entered.

Caution:

When censorship has been enabled, the only way to regain access is with the password. If this is forgotten or not correctly configured, then there is no way back into the device.

There are two 64-bit values stored in the shadow flash which control the censorship (see *Table 172* for a full description):

- Nonvolatile Private Censorship Password registers, NVPWD0 and NVPWD1
- Nonvolatile System Censorship Control registers, NVSCI0 and NVSCI1

Censorship password registers (NVPWD0 and NVPWD1)

The two private password registers combine to form a 64-bit password that should be programmed to a value known only by you. After factory test these registers are programmed as shown below:

- NVPWD0 = 0xFEED_FACE
- NVPWD1 = 0xCAFE_BEEF

This means that even if censorship was inadvertently enabled by writing to the censorship control registers, there is an opportunity to get back into the microcontroller using the default private password of 0xFEED_FACE_CAFE_BEEF.

When configuring the private password, each half word (16-bit) must contain at least one "1" and one "0". Some examples of legal and illegal passwords are shown in *Table 615*:

Table 615. Examples of legal and illegal passwords

Legal (valid) passwords	Illegal (invalid) passwords
0x0001_0001_0001_0001 0xFFFE_FFFE_FFFE_FFFE 0x1XXX_X2XX_XX4X_XXX8	0x0000_XXXX_XXXX_XXXX 0xFFFF_XXXX_XXXX_XXXX

In uncensored devices it is possible to download code via LINFlex or FlexCAN (Serial Boot Mode) into internal SRAM even if the 64-bit private password stored in the flash and provided during the boot sequence is a password that does not conform to the password rules.

Nonvolatile System Censorship Control registers (NVSCI0 and NVSCI1)

These registers are used together to define the censorship configuration. After factory test these registers are programmed as shown below which disables censorship:

- NVSCI0 = 0x55AA_55AA
- NVSCI1 = 0x55AA_55AA

Each 32-bit register is split into an upper and lower 16-bit field. The upper 16 bits (the SC field) are used to control serial boot mode censorship. The lower 16 bits (the CW field) are used to control flash memory boot censorship.

Caution:

If the contents of the shadow flash memory are erased and the NVSCI0,1 registers are not re-programmed to a valid value, the microcontroller will be permanently censored with no way for you to regain access. A microcontroller in this state cannot be debugged or re-flashed.

Censorship configuration

The steps to configuring censorship are:

1. Define a valid 64-bit password that conforms to the password rules.
2. Using the table and flow charts below, decide what level of censorship you require and configure the NVSCI0,1 values.
3. Re-program the shadow flash memory and NVPWD0,1 and NVSCI0,1 registers with your new values. A POR is required before these will take effect.

Caution:

If
(NVSCI0 and NVSCI1 do not match)
or
(Either NVSCI0 or NVSCI1 is not set to 0x55AA)
then the microcontroller will be permanently censored with no way to get back in.

Table 616 shows all the possible modes of censorship. The red shaded areas are to be avoided as these show the configuration for a device that is permanently locked out. If you wish to enable censorship with a private password there is only one valid configuration — to modify the CW field in both NVSCI0,1 registers so they match but do not equal 0x55AA. This will allow you to enter the private password in both serial and flash boot modes.

Table 616. Censorship configuration and truth table

<table>
<tr><th colspan="2">Boot configuration</th><th rowspan="2">Serial censorship control word (NVSCIn[SC])</th><th rowspan="2">Censorship control word (NVSCIn[CW])</th><th rowspan="2">Internal flash memory state</th><th rowspan="2">Nexus state</th><th rowspan="2">Serial password</th><th rowspan="2">JTAG password</th></tr>
<tr><th>FAB pin state</th><th>Control options</th></tr>
<tr><td rowspan="4">0 (flash memory boot)</td><td>Uncensored</td><td>0xXXXX AND NVSCI0 == NVSCI1</td><td>0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Enabled</td><td rowspan="4"></td><td>N/A</td></tr>
<tr><td>Private flash memory password and censored</td><td>0x55AA AND NVSCI0 == NVSCI1</td><td>!0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Enabled with password</td><td>NVPWD1,0 (SSCM reads flash memory(1))</td></tr>
<tr><td rowspan="2">Censored with no password access (lockout)</td><td>!0x55AA</td><td>!0X55AA</td><td rowspan="2">Enabled</td><td rowspan="2">Disabled</td><td rowspan="2">N/A</td></tr>
<tr><td colspan="2">OR NVSCI0 != NVSCI1</td></tr>
<tr><td rowspan="5">1 (serial boot)</td><td>Private flash memory password and uncensored</td><td colspan="2">0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Enabled</td><td>NVPWD0,1 (BAM reads flash memory(1))</td><td rowspan="4"></td></tr>
<tr><td>Private flash memory password and censored</td><td>0x55AA AND NVSCI0 == NVSCI1</td><td>!0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Disabled</td><td>NVPWD1,0 (SSCM reads flash memory(1))</td></tr>
<tr><td>Public password and uncensored</td><td>!0x55AA AND NVSCI0 != NVSCI1</td><td>0X55AA AND NVSCI0 != NVSCI1</td><td>Enabled</td><td>Enabled</td><td>Public (0xFEED_FACE_CAFE_BEEF)</td></tr>
<tr><td rowspan="2">Public password and censored (lockout)</td><td colspan="2">!0x55AA</td><td rowspan="2">Disabled</td><td rowspan="2">Disabled</td><td rowspan="2">Public (0xFEED_FACE_CAFE_BEEF)</td></tr>
<tr><td colspan="2">OR NVSCI0 != NVSCI1</td></tr>
</table>

= Microcontroller permanently locked out

= Not applicable

1. When the SSCM reads the passwords from flash memory, the NVPWD0 and NVPWD1 password order is swapped, so you have to submit the 64-bit password as {NVPWD1, NVPWD0}.

The flow charts in *Figure 654* and *Figure* provide a way to quickly check what will happen with different configurations of the NVSCI0,1 registers as well as detailing the correct way to enter the serial password. In the password examples, assume the 64-bit password has been

programmed into the shadow flash memory in the order {NVPWD0, NWPWD1} and has a value of 0x01234567_89ABCDEF.

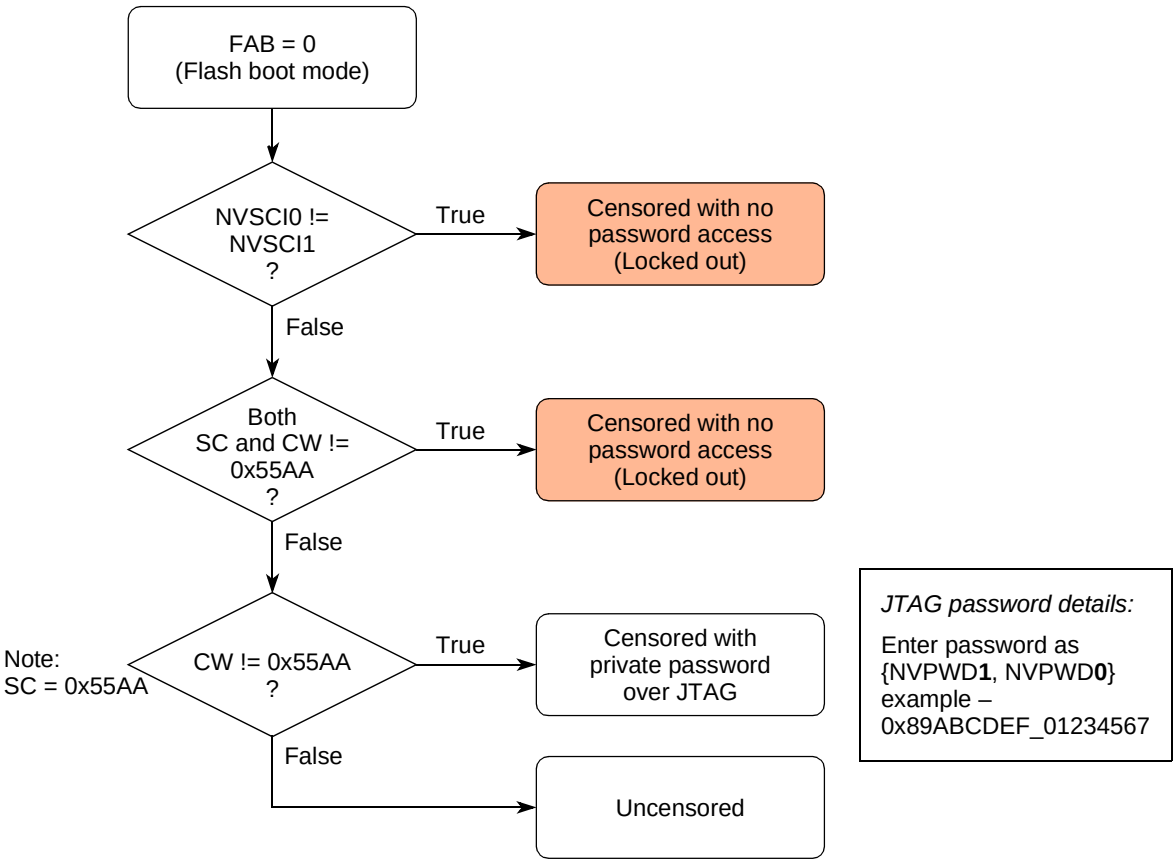
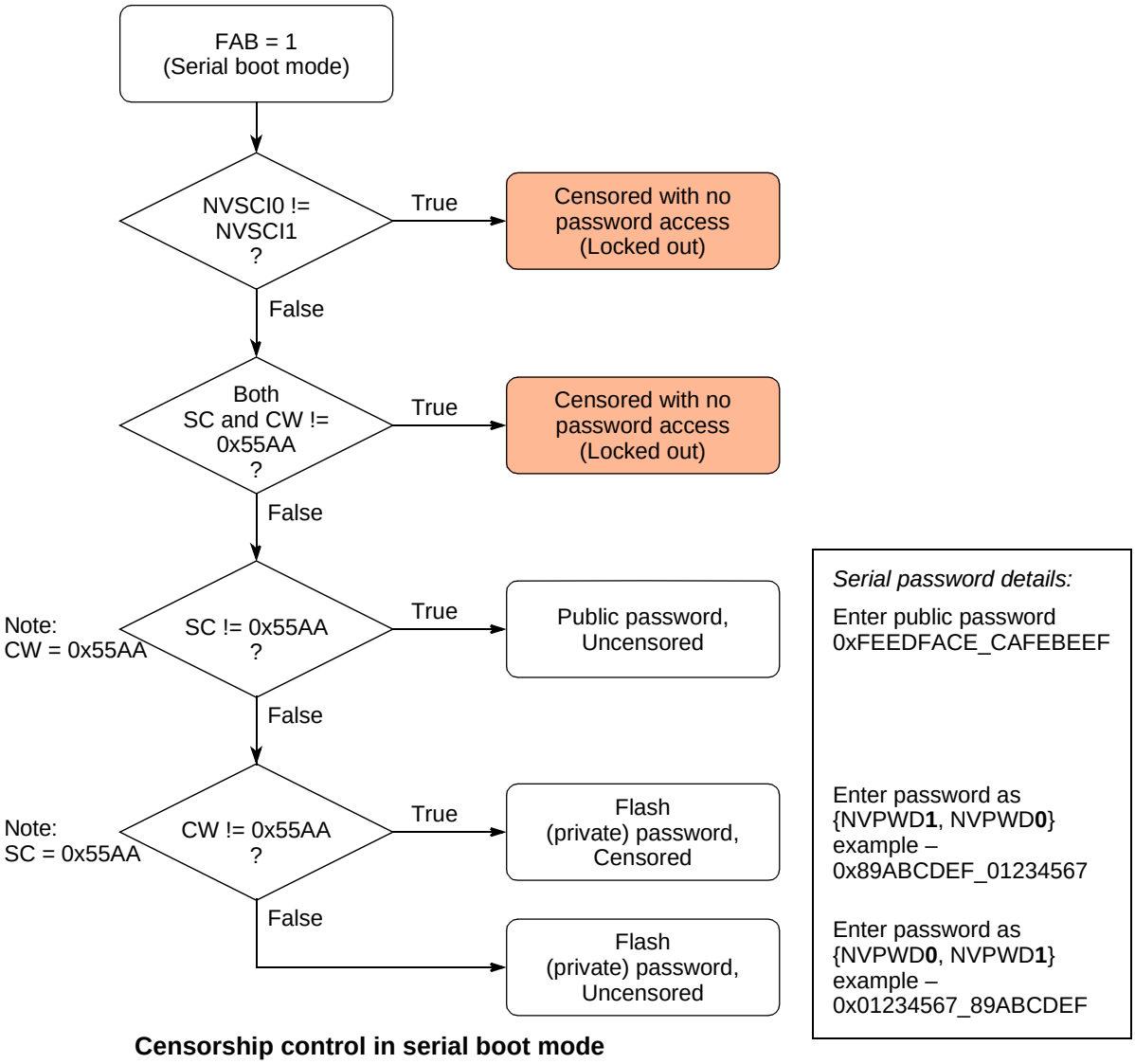


Figure 654. Censorship control in flash memory boot mode



Censorship control in serial boot mode

36 Voltage Regulators and Power Supplies

36.1 Voltage regulator

The power blocks are used for providing 1.2 V digital supply to the internal logic of the device. The main/input supply is 3.3 V to 5.0 V ±10% and the digital/regulated output supply has a trim target voltage of 1.28 V. The voltage regulator used in SPC56xP60/54 is the high power or main regulator (HPREG).

The internal voltage regulator requires an external ballast transistor and (external) capacitance (CREG) to be connected to the device in order to provide a stable low voltage digital supply to the device. Capacitances should be placed on the board as near as possible from the associated pins.The regulator has a digital domain called the High Power domain that has a low voltage detector for the 1.2 V output voltage. Additionally, there are two low voltage detectors for the main/input supply with different threshold, one at 3.3 V level and the other one at 5 V level.

36.1.1 High Power or Main Regulator (HPREG)

The HPREG converts the 3.3 V–5 V input supply to a 1.2 V digital supply. The nominal target output is 1.28 V. Due to all variations, the actual output will be in range of 1.08 V to 1.32 V in the full current load range (0–250 mA) after trimming.

The stabilization for HPREG is achieved using an external capacitance. The minimum recommended value is 3 × 10 µF with low ESR (refer to datasheet for details).

Note: In general an offset voltage must be avoided to pre-charge $V_{DD\_HV\_REG}$ through parasitic paths to allow a correct power up sequence.

The MCU supply must power on from GND to power supply with a monotonic ramp, minimum and maximum values as described in the data sheet (TV_{DD}).

36.1.2 Low Voltage Detectors (LVD) and Power On Reset (POR)

Five types of low voltage detectors are provided on the device:

- $V_{REGLVDMOK\_L}$ monitors the 3.3 V regulator supply
- $V_{FLLVDMOK\_L}$ monitors the 3.3 V flash supply
- $V_{IOLVDMOK\_L}$ monitors the 3.3 V I/O supply
- $V_{IOLVDM5OK\_L}$ monitors the 5 V I/O supply
- $V_{MLVDDOK\_L}$ monitors the 1.2 V digital logic supply

LVD_MAIN is the main voltage LVD with a threshold set around 2.7 V. LVD_MAIN5 is the main voltage LVD with a threshold set around 4 V. The LVD_MAIN and LVD_MAIN5 sense the 3.3 V–5 V power supply for CORE, shared with IO ring supply and indicate when the 3.3 V–5 V supply is stabilized. The threshold levels of LVD_MAIN5 are trimmable with the help of LVDM5[0:3] trim bits.

The LVD_MAIN and LVD_MAIN5 detectors sense the V_{DDIO} supply and provide $V_{IOLVDMOK\_H}$/$V_{IOLVDM5OK\_H}$ and $V_{IOLVDMOK\_L}$/$V_{IOLVDM5OK\_L}$ as active high signals at 3.3 V and 1.2 V supply levels, respectively.

Two more LVD_MAIN detectors are also used for sensing VDDREG and VDDFLASH.

An LVD_DIG in the regulator senses the HPREG output. It provides $V_{MLVDDOK\_H}$ and V_{MLVDOK_L} as active high signals.

The reference voltage used for all LVDs is trimmed for LVD_DIG using the bits LP[4:7]. Therefore, during the pre-trimming period, LVD_DIG exhibits higher thresholds, whereas post trimming, the thresholds come in the desired range. The trimming bits are provided by SSCM device option bits, which are updated during the reset phase (RGM reset phase 2) only. Power-down pins are provided for LVDs. When LVDs are powered down, their outputs are pulled high. LVDs are not controllable by the user. The only option is the possibility to mask LVD_MAIN5 using the mask bit (5V_LVD_MASK) in the VREG_CTL register.

POR is required to initialize the device during supply rise. POR works only on the rising edge of main supply. To ensure its functioning during the following rising edge of the supply, it is reset by the output of the LVD_MAIN block when main supply reaches below the lower voltage threshold of the LVD_MAIN.

POR is asserted on power-up when V_{DD} supply is above V_{PORUP} minimum (refer to datasheet for details). It is released only after V_{DD} supply is above V_{PORH} (refer to datasheet for details). V_{DD} above V_{PORH} ensures power management module including internal LVDs modules are fully functional.

36.1.3 VREG digital interface

The voltage regulator digital interface provides the temporization delay at initial power-up and at exit from low-power modes. A signal, indicating that Low Power domain is powered, is used at power-up to release reset to temporization counter. On completion of the delay counter, a end-of-count signal is released, it is gated with an other signal indicating main domain voltage fine in order to release the VREGOK signal. This is used by RGM to release the reset to the device.

The VREG digital interface also holds control register to mask 5 V LVD status coming from the voltage regulator at the power-up.

36.1.4 Registers Description

Voltage Regulator Control Register (VREG_CTL)

Figure 655. Voltage Regulator Control register (VREG_CTL)

Address: 0xC3FE_8080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V_LVD_MASK
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Table 617. VREG_CTL field descriptions

Field	Description
5V_LVD_MASK	Mask bit for 5 V LVD from regulator This is a read/write bit and must be unmasked by writing a 1 by software to generate LVD functional reset request to RGM for 5V trip. 05 V LVD not masked 15 V LVD masked

Voltage Regulator Status register (VREG_STATUS)

Figure 656. Voltage Regulator Status register (VREG_STATUS)

Address: 0xC3FE_8084 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V_LVD_STATUS
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Table 618. VREG_STATUS field descriptions

Field	Description
5V_LVD_STATUS	Status bit for 5 V LVD from regulator 05 V LVD not OK 15 V LVD OK

36.2 Power supply strategy

The SPC56xP60/54 provides three dedicated supply domains at the package level:

- HV—High voltage external power supply for I/Os, voltage regulator module, and most analog modules

  This must be provided externally through V_{DD_HV}/V_{SS_HV} power pins. Voltage values should be aligned with V_{DD}/V_{SS}. Refer to the device datasheet for details.
- ADC—High voltage external power supply for ADC module

  This must be provided externally through $V_{DD\_HV\_ADx}/V_{SS\_HV\_ADx}$ power pins. Voltage values should be aligned with $V_{DD\_HV\_ADx}/V_{SS\_HV\_ADx}$. Refer to the device datasheet for details.
- LV—Low voltage internal power supply for core, PLL, and flash digital logic

  This is provided to the core, PLL and flash. Four V_{DD_LV}/V_{SS_LV} pins pairs are provided to connect the low voltage power supply. Refer to the device datasheet for details.

The three dedicated supply domains are further divided within the package in order to reduce EMI and noise as much as possible:

- HV_REG—High voltage regulator supply
- HV_IO*n*—High voltage PAD supply
- HV_FL—High voltage flash supply
- HV_OSC(n)—High voltage external oscillator and regulator supply
- HV_AD—High voltage supply and reference for ADC module. Supplies are further star routed to reduce impact of ADC resistive reference on ADC capacitive reference accuracy.
- LV_COR*n*—Low voltage supply for the core. It is also used to provide supply for PLL and Flsah memory through double bonding.

n.Regulator ground is separated from oscillator ground and shorted to the LV ground through star routing.

37 IEEE 1149.1 Test Access Port Controller (JTAGC)

37.1 Introduction

The JTAG port of the device consists of three inputs and one output. These pins include test data input (TDI), test mode select (TMS), test clock input (TCK) and test data output (TDO). TDI, TMS, TCK and TDO are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

37.2 Block diagram

Figure 657 is a block diagram of the JTAG Controller (JTAGC).

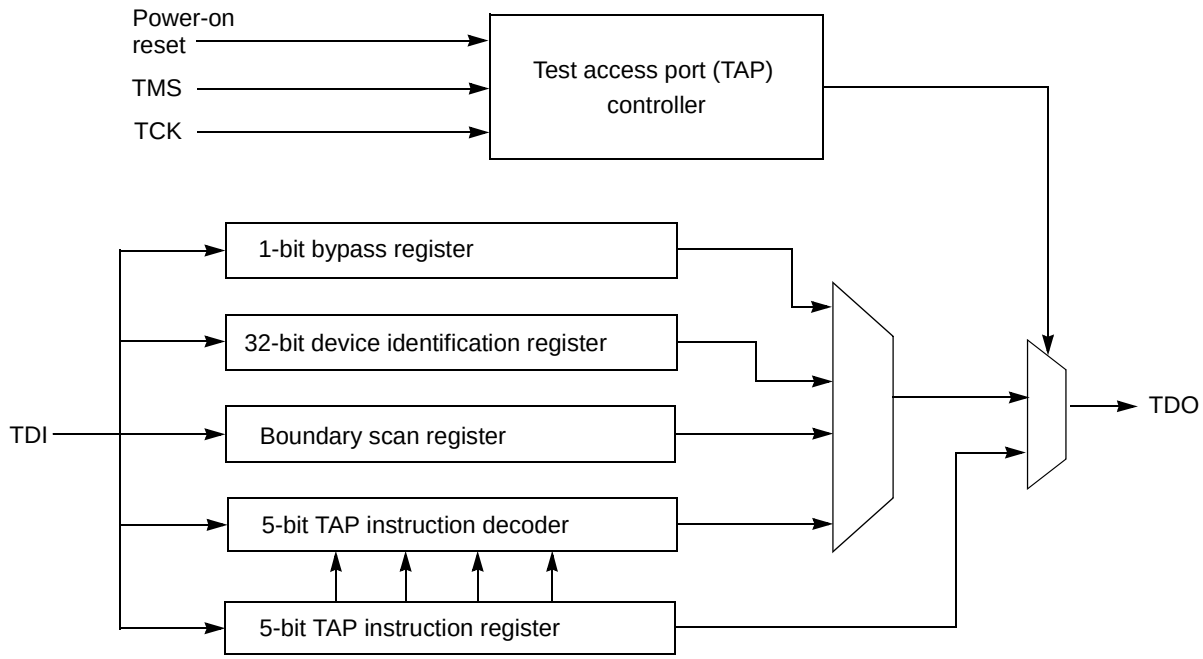


Figure 657. JTAG controller block diagram

37.3 Overview

The JTAGC provides the means to test device functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

37.4 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
- Four pins (TDI, TMS, TCK, and TDO)—see *Section 37.6: External signal description.*
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions.
- Test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry.

37.5 Modes of operation

The JTAGC uses a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

37.5.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

37.5.2 IEEE 1149.1-2001 defined test modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data registers that can operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in *Section 37.8.4: JTAGC instructions.*

Bypass mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register provides a minimum-length serial path to shift data between TDI and TDO.

TAP sharing mode

There are four selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC), e200 OnCE, and eDMA Nexus. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_CORE0, ACCESS_AUX_TAP_CORE1, ACCESS_AUX_TAP_NASPS_0, and ACCESS_AUX_TAP_NASPS_1 and ACCESS_AUX_TAP_NPC. Instruction opcodes for each instruction are shown in *Table 621*.

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers refer to *Chapter 38: Nexus Port Controller (NPC).*

37.6 External signal description

The JTAGC consists of four signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in *Table 619*.

Table 619. JTAG signal properties(1)

Name	I/O	Function	Reset state	Pull(2)
TCK	I	Test clock	—	Down
TDI	I	Test data in	—	Up
TDO	O	Test data out	High Z(3)	Down(3)
TMS	I	Test mode select	—	Up

1. Test clock frequency must always be less than one fourth of system clock frequency.
2. The pull is not implemented in this module. Pullup/down devices are implemented in the pads.
3. TDO output buffer enable is negated when JTAGC is not in the Shift-IR or Shift-DR states. A weak pulldown can be implemented on TDO.

37.7 Memory map and registers description

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

37.7.1 Instruction register

The JTAGC uses a 5-bit instruction register as shown in *Figure 658*. The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

Figure 658. 5-bit Instruction register

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset	0	0	0	0	1

37.7.2 Bypass register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

37.7.3 Device identification register

The device identification register, shown in *Figure 659*, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

Figure 659. Device identification register

IR[4:0]: 0_0001 (IDCODE) Access: User read-only

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
R	PRN	DC		PIN		MIC		ID
W								
Reset	0 0 0 0	1 0 1 0	1 1 1 0	0 0 1 0	0 1 0 0	0 0 0 0	0 1 0 0	0 0 0 1

Table 620. Device identification register field descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. For the SPC56xP60/54 this value is 0x2B.
10–19 PIN	Part identification number. Contains the part number of the device. For the SPC56xP60/54, this value is 0x224.
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for STMicroelectronics, 0x20.
31 ID	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

37.7.4 Boundary scan register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in *Section 37.8.5: Boundary scan.* The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file.

37.8 Functional description

37.8.1 JTAGC reset configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

37.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port (TAP)

The JTAGC uses the IEEE 1149.1-2001 Test Access Port (TAP) for accessing registers. This port can be shared with other TAP controllers on the MCU. For more detail on TAP sharing via JTAGC instructions refer to *Section : ACCESS_AUX_TAP_x instructions.*

Data is shifted between TDI and TDO though the selected register starting with the least significant bit, as illustrated in *Figure 660*. This applies for the instruction register, test data registers, and the bypass register.

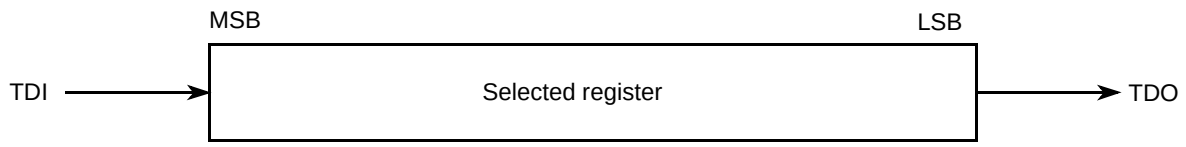
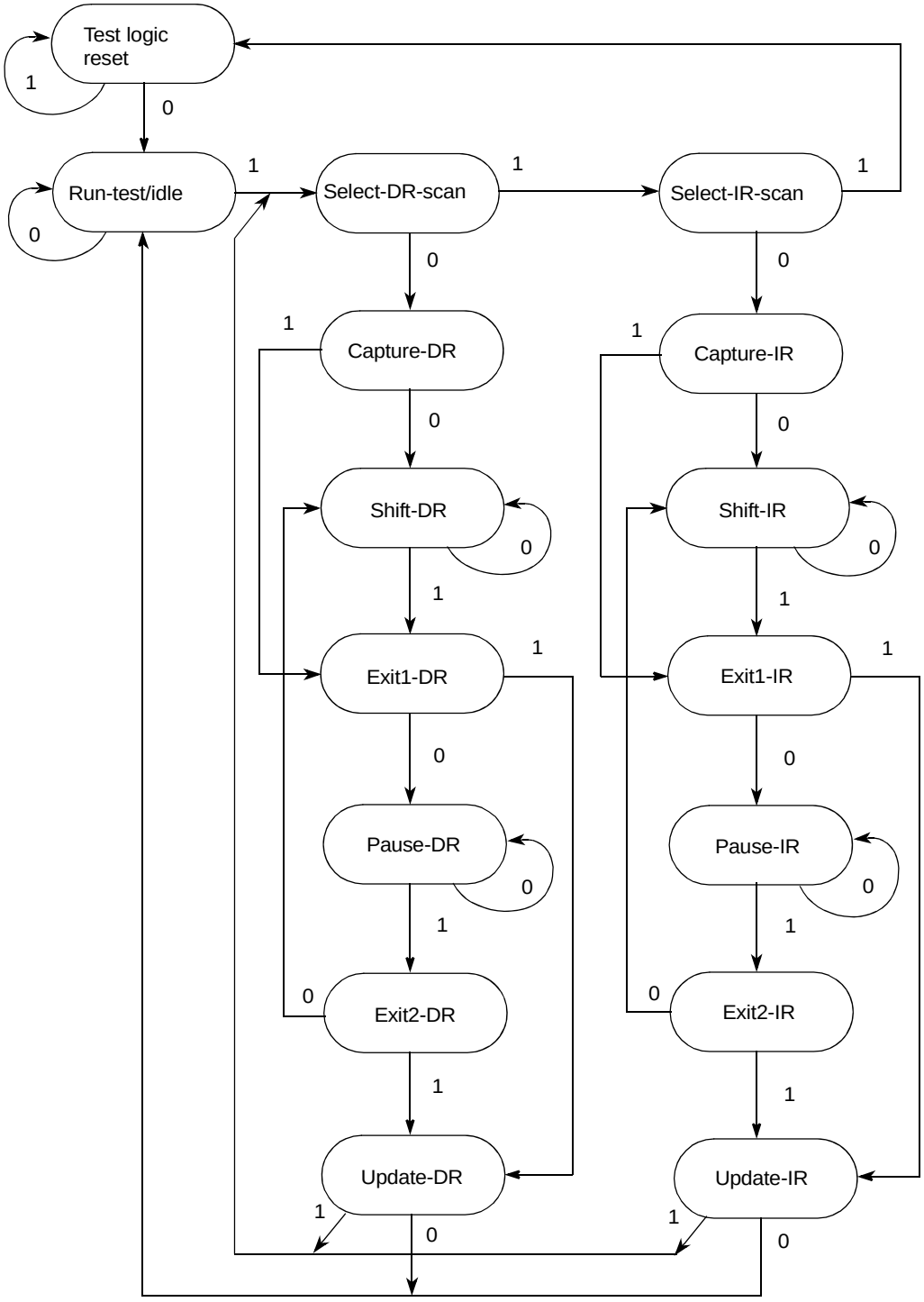


Figure 660. Shifting data through a register

37.8.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 661* shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal.

As *Figure 661* shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 661. IEEE 1149.1-2001 TAP controller finite state machine

Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path reads or writes the register data by shifting in the data (LSB first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after fetching the required number of bits.

37.8.4 JTAGC instructions

This section gives an overview of each instruction. Refer to the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in *Table 621*.

Table 621. JTAG instructions

Instruction	Code[4:0]	Instruction summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
HIGHZ	01001	Selects bypass register while tristating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
ACCESS_AUX_TAP_CORE0	10001	Enables access to Core_0 TAP controller
ACCESS_AUX_TAP_CORE1	11001	Enables access to Core_1 TAP controller
ACCESS_AUX_TAP_NASPS_0	10111	Selects the ACCESS_AUX_NASPS_0 configuration that connects the auxiliary TAP interface to the Nexus SRAM Port sniffer connected between the XBAR_0 and the SRAMC_0
ACCESS_AUX_TAP_NASPS_1	11000	Selects the ACCESS_AUX_NASPS_1 configuration that connects the auxiliary TAP interface to the Nexus SRAM Port sniffer connected between the XBAR_0 and the SRAMC_1
BYPASS	11111	Selects bypass register for data operations

Table 621. JTAG instructions (continued)

Instruction	Code[4:0]	Instruction summary
Factory Debug Reserved(1)	00101 00110 01010	Intended for factory debug only
Reserved	All Other Codes	Decoded to select bypass register

1. Intended for factory debug, and not customer use.

BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

ACCESS_AUX_TAP_*x* instructions

The ACCESS_AUX_TAP_*x* instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

CLAMP instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

EXTEST — external test instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

HIGHZ instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active, all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

IDCODE instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

SAMPLE instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising-edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- The PRELOAD part of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

37.8.5 Boundary scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

37.9 e200z0 OnCE controller

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug features, as well as providing access to the Nexus 2+ configuration registers. A complete discussion of the e200z0 OnCE debug features is available in the core reference manual.

37.9.1 e200z0 OnCE controller block diagram

Figure 662 is a block diagram of the e200z0 OnCE block.

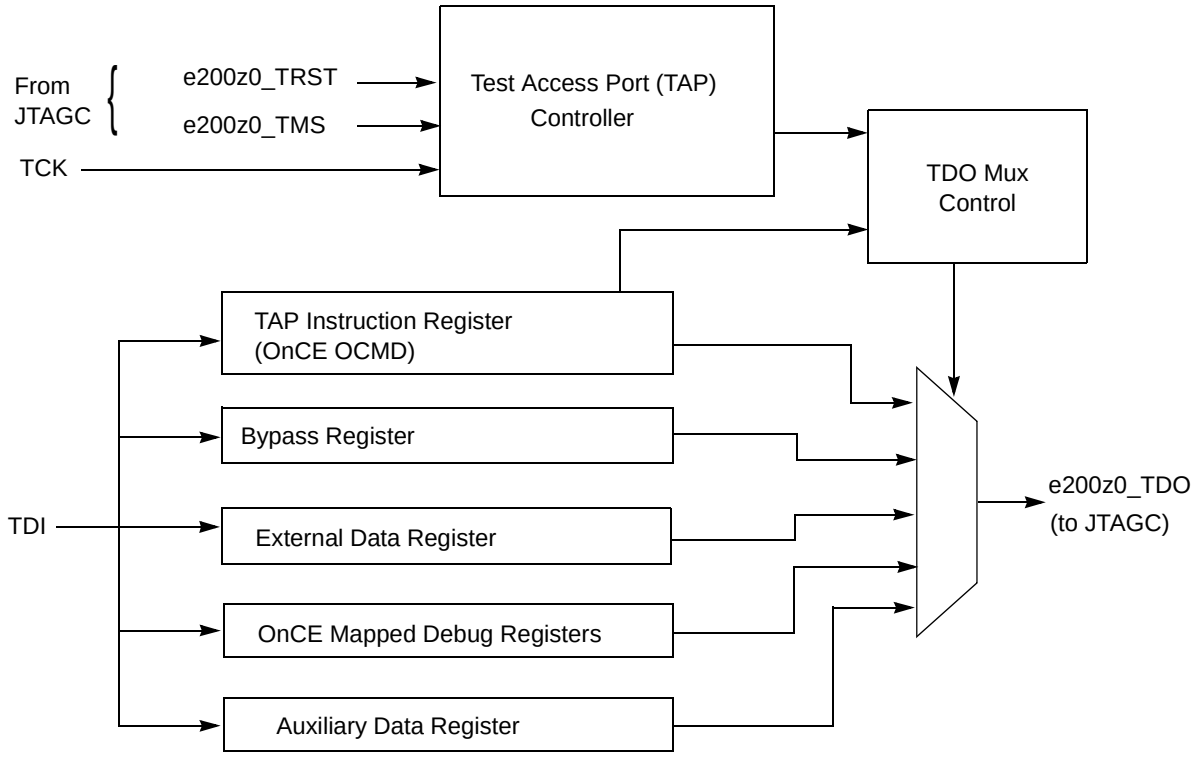


Figure 662. e200z0 OnCE block diagram

37.9.2 e200z0 OnCE controller functional description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described as follows.

Enabling the TAP controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in *Section : TAP sharing mode.* The e200z0 OnCE TAP controller may either be accessed independently or chained with the e200z1 OnCE TAP controller, such that the TDO output of the e200z1 TAP controller is fed into the TDI input of the e200z0 TAP controller. The chained configuration allows commands to be loaded into both core's OnCE registers in one shift operation, so that both cores can be sent a GO command at the same time for example.

37.9.3 e200z0 OnCE controller registers description

Most e200z0 OnCE debug registers are fully documented in the core reference manual.

OnCE Command register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in *Figure 663*. The OCMD is updated when the TAP controller enters the update-IR state. It contains fields for controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

Figure 663. OnCE Command register (OCMD)

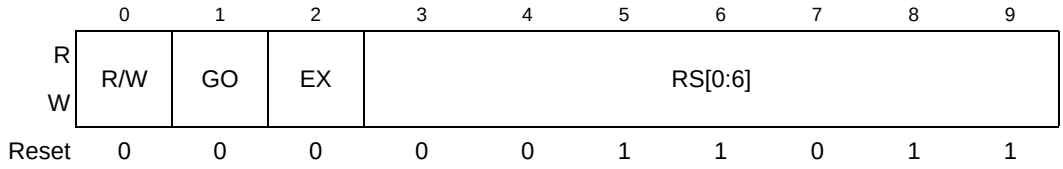


Table 622. e200z0 OnCE register addressing

RS[0:6]	Register selected
000 0000 000 0001	Reserved
000 0010	JTAG ID (read-only)
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110	Data Value Compare 1 (DVC1)
010 0111	Data Value Compare 2 (DVC2)
010 1000 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)

Table 622. e200z0 OnCE register addressing (continued)

RS[0:6]	Register selected
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)
011 0100 – 101 1111	Reserved (do not access)
110 1111	Shared Nexus Control Register (SNC) (only available on the e200z0 core)
111 0000 – 111 1001	General Purpose Register Selects [0:9]
111 1010 – 111 1011	Reserved
111 1100	Nexus 2+ Access
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

37.10 Initialization/Application Information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Place the JTAGC in reset through TAP controller state machine transitions controlled by TMS
   – Load the appropriate instruction for the test or action to be performed.

38 Nexus Port Controller (NPC)

38.1 Information specific to this device

This section presents device-specific parameterization, customization, and feature availability information not specifically referenced in the remainder of this chapter.

38.1.1 Parameter values

The parameter values for this device are shown in *Table 623*.

Table 623. Device parameter values

Parameter	Value
Number of MDO pins available in reduced-port mode	4
Number of MDO pins available in full-port mode	12
Number of blocks that input an $\overline{\text{EVTO}}$	1
Part identification number	0x2A2
Manufacturer identity code (MIC)	0x020
Design center code (DC)	0x2B

38.1.2 Unavailable features

This device does not support an MCKO division factor of 3. Therefore, the setting PCR[MCKO_DIV] = 2 (see *Table 628*) is not available on this device.

Nexus DDR mode works only for MCKO division factors of 2, 4, and 8. Therefore, the setting PCR[MCKO_DIV] = 0 (see *Table 628*) will not support Nexus DDR mode on this device.

38.2 Introduction

Figure 664 is a block diagram of the Nexus Port Controller (NPC) block.

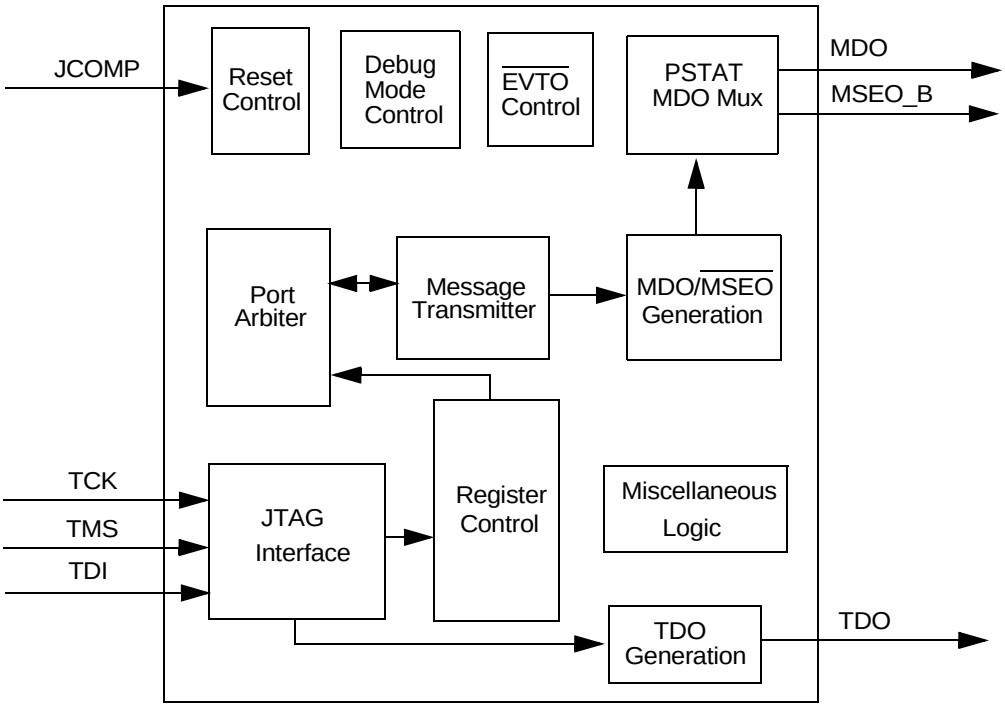


Figure 664. Nexus Port Controller Block Diagram

38.2.1 Overview

On a system-on-a-chip device, there are often multiple blocks that require development support. Each of these blocks implements a development interface based on the IEEE-ISTO 5001-2001 standard. The blocks share input and output ports that interface with the development tool. The NPC controls the usage of the input and output port in a manner that allows all the individual development interface blocks to share the port, and appear to the development tool to be a single block.

38.2.2 Features

The NPC block performs the following functions:

- Controls arbitration for ownership of the Nexus Auxiliary Output Port
- Nexus Device Identification Register and Messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$
- Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle
- Control of the device-wide debug mode
- Generates censorship status, and power-on reset status
- System clock locked status indication via MDO[0] following power-on reset
- Provides Nexus support for censorship mode
- $\overline{\text{RDY}}$ pin support to increase the transfer rate of the IEEE 1149.1 port for large data read requests

38.2.3 Modes of operation

The NPC block uses the JCOMP input, the censorship status , and an internal power-on reset indication as its primary reset signals. Upon exit of reset, the mode of operation is determined by the Port Configuration Register (PCR) settings.

Reset

The NPC block is asynchronously placed in reset when either power-on reset is asserted, JCOMP is not set for Nexus access, the device enters censored mode , or the TAP controller state machine is in the Test-Logic-Reset state. Holding TMS high for 5 consecutive rising edges of TCK guarantees entry into the Test-Logic-Reset state regardless of the current TAP controller state. Following negation of power-on reset, the NPC remains in reset until the system clock achieves lock. The NPC is unaffected by other sources of reset. While in reset, the following actions occur:

- The TAP controller is forced into the Test-Logic-Reset state
- The auxiliary output port pins are negated
- The TDI, TMS, and TCK TAP inputs are ignored (when in power-on reset, censored mode or JCOMP not set for NPC operation only)
- Registers default back to their reset values

Disabled-Port Mode

In disabled-port mode, auxiliary output pin port enable signals are negated, thereby disabling message transmission. Any debug feature that generates messages can not be used. The primary features available are class 1 features and read/write access to the registers. Class 1 features include the ability to trigger a breakpoint event indication through EVTO.

Full-Port Mode

Full-port mode (FPM) is entered by asserting the MCKO_EN and FPM bits in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. The number of MDO pins available is device-specific.

Reduced-Port Mode

Reduced-port mode (RPM) is entered by asserting the MCKO_EN bit and deasserting the FPM bit in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. The number of MDO pins available is device-specific.

Censored Mode

When the device is in censored mode, reading the contents of internal flash externally is not allowed. To prevent Nexus modules from violating censorship, the NPC is held in reset when in censored mode, asynchronously holding all other Nexus modules in reset as well. This prevents Nexus read/write to memory mapped resources and the transmission of Nexus trace messages.

Nexus Double Data Rate Mode

Nexus double data rate (DDR) mode is enabled by asserting the DDR_EN bit in the PCR. In double data rate mode, message data is updated between the edges (both rising and falling, for cut2) of MCKO, effectively doubling message throughput.

38.3 External signal description

38.3.1 Overview

The NPC pin interface provides for the transmission of messages from Nexus blocks to the external development tools and for access to Nexus client registers. The NPC pin definition is outlined in *Table 624*.

Table 624. NPC Signal Properties

Name	Port	Function	Reset State	Pull(1)
$\overline{\text{EVTO}}$	Auxiliary	Event Out pin	0b1	—
JCOMP	JTAG	JTAG Compliancy and TAP Sharing Control	—	Down
MDO	Auxiliary	Message Data Out pins	0(2)	—
MSEO	Auxiliary	Message Start/End Out pins	0b11	—
TCK	JTAG	Test Clock Input	—	Down
TDI	JTAG	Test Data Input	—	Up
TDO	JTAG	Test Data Output	High Z(3)	—
TMS	JTAG	Test Mode Select Input	—	Up
$\overline{\text{RDY}}$	JTAG	Data ready for transfer to/from NRRs	—	—

1. The pull is not implemented in this block. Pullup/pulldown devices are implemented in the pads.
2. Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.
3. TDO output buffer enable is negated when the NPC is not in the Shift-IR or Shift-DR states. A weak pull may be implemented on TDO at the SoC level.

38.3.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 624* in more detail. The JTAG test clock (TCK) input from the pin is not a direct input to the NPC. The NPC requires two separate input clocks for TCK clocked logic, one for posedge (rising edge TCK) logic and one for negedge (falling edge TCK) logic. Both clocks are derived from the pin TCK, and generated external to the NPC.

$\overline{\text{EVTO}}$ - Event Out

Event Out ($\overline{\text{EVTO}}$) is an output pin that is asserted upon breakpoint occurrence to provide breakpoint status indication. The $\overline{\text{EVTO}}$ output of the NPC is generated based on the values of the individual $\overline{\text{EVTO}}$ signals from all Nexus blocks that implement the signal.

JCOMP - JTAG Compliancy

The JCOMP signal provides the ability to share the TAP. The NPC TAP controller is enabled when JCOMP is set to the NPC enable encoding, otherwise the NPC TAP controller remains in reset.

MDO - Message Data Out

Message Data Out (MDO) are output pins used for uploading OTM, BTM, DTM, and other messages to the development tool. The development tool should sample MDO on the rising edge of MCKO. The width of the MDO bus used is determined by reset configuration.

MSEO_B - Message Start/End Out

Message Start/End Out ($\overline{\text{MSEO}}$) are two output pins that indicates when a message on the MDO pins has started, when a variable length packet has ended, or when the message has ended. The development tool should sample $\overline{\text{MSEO}}$ on the rising edge of MCKO.

TCK - Test Clock Input

Test Clock Input (TCK) pin is used to synchronize the test logic and control register access through the JTAG port.

TDI - Test Data Input

Test Data Input (TDI) pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

TDO - Nexus Test Data Output

Test Data Output (TDO) pin transmits serial output for instructions and data. TDO is three-stateable and is actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO is updated on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

TMS - Test Mode Select

Test Mode Select Input (TMS) pin is used to sequence the IEEE 1149.1-2001 TAP controller state machine. TMS is sampled on the rising edge of TCK.

$\overline{\text{RDY}}$ — Data ready for transfer

The $\overline{\text{RDY}}$ pin, supported on cut2 only, exists to increase the transfer rate of the IEEE 1149.1 port. It is used to signal when data are ready to be transferred to and from NRRs. This may eliminate the need to poll NRRs for status information for synchronization purposes. This capability becomes especially important when performing read/write access transfers to different speed target memories.

The $\overline{\text{RDY}}$ pin asserts (asynchronously) a logic low whenever the read/write access transfer has completed without error and then deasserts when the IEEE 1149.1 state machine has reached the CAPTURE_DR state.

38.4 Register definition

This section provides a detailed description of the NPC registers accessible to the end user. Individual bit-level descriptions and reset states of the registers are included.

Table 625 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC block does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass and instruction registers have no index values. These registers are not accessed in the same manner as Nexus client registers. Refer to the individual register descriptions for more detail.

Table 625. NPC Registers

Index	Register
0	Device ID Register (DID)
127	Port Configuration Register (PCR)

38.4.1 Register descriptions

This section consists of NPC register descriptions.

Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

Instruction Register

The NPC block uses a 4-bit instruction register as shown in *Table 665*. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the block to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and Test-Logic-Reset TAP controller states. Synchronous entry into the Test-Logic-Reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the Test-Logic-Reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

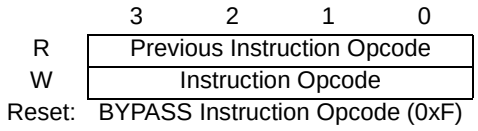


Figure 665. 4-Bit Instruction Register

Nexus Device ID Register (DID)

The device identification register, shown in *Figure 666*, allows the part revision number, design center, part identification number, and manufacturer identity code of the part to be determined through the auxiliary output port.

Register index: 0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Part Revision Number				Design Center						Part Identification Number					
W																
RESET:	PRN				DC						PIN					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Part Identification Number				Manufacturer Identity Code											1
W																
RESET:	PIN (cont'd.)				0	0	0	0	0	0	0	1	1	1	0	1

= Reserved

Figure 666. Nexus Device ID Register

Table 626. DID Field Descriptions

Bit	Name	Description
31:28	PRN	Part Revision Number These bits contain the revision number of the part
27:22	DC	Design Center These bits indicate the device design center
21:12	PIN	Part Identification Number These bits contain the part number of the device
11:1	MIC	Manufacturer Identity Code These bits contain the reduced Joint Electron Device Engineering Council (JEDEC) ID
0	Bit [0]	IDCODE Register ID Bit [0] identifies this register as the device identification register and not the bypass register

Port Configuration Register (PCR)

Register index: 127

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	DDR_EN	NEXCFG	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LP_DBG	0	0	0	0	0	LP2_SYN	LP1_SYN	0	0	0	0	0	0	0	PSTAT_EN
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Reserved

Figure 667. Port Configuration Register (PCR)

The PCR, shown in *Figure 667*, is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NPC is enabled.

The PCR register may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP_DBG and LPn_SYN bits, but must preserve the original state of the remaining bits in the register.

Note: The mode or clock division must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Table 627. PCR field descriptions

Name	Description
FPM	Full Port Mode The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 1 = All MDO pins are used to transmit messages 0 = A subset of MDO pins are used to transmit messages
MCKO_GT	MCKO Clock Gating Control This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 1 = MCKO gating is enabled 0 = MCKO gating is disabled
MCKO_EN	MCKO Enable This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 1 = MCKO clock is enabled 0 = MCKO clock is driven to zero

Table 627. PCR field descriptions (continued)

Name	Description
MCKO_DIV	MCKO Division Factor The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. *Table 628* shows the meaning of MCKO_DIV Values. In this table, SYS_CLK represents the system clock frequency.
EVT_EN	EVTO/EVTI Enable This bit enables the EVTO/EVTI port functions. 1 = EVTO/EVTI port enabled 0 = EVTO/EVTI port disabled
DDR_EN	Double Data Rate Mode Enable This bit enables Nexus double data rate (DDR) mode. In DDR mode, message data is updated on both rising and falling edges of MCKO, effectively doubling message throughput. 1 = DDR mode enabled 0 = DDR mode disabled
NEXCFG	Nexus Configuration Select Generic Nexus control bit. Function is SoC specific. 1 = NEXCFG set 0 = NEXCFG cleared
LP_DBG_EN	Low Power Debug Enable This bit enables debug functionality on exit from low power modes on supported devices. 1 = Low power debug enabled 0 = Low power debug disabled
LPn_SYN	Low Power Mode n Synchronization These bits are used to synchronize the entry into low power modes between the device and debug tool. Supported devices set these bits before a pending entry into low power mode. After reading the bit as set, the debug tool then clears the bit to acknowledge to the device that it may enter the low power mode. 1 = Low power mode entry pending 0 = Low power mode entry acknowledged
PSTAT_EN	Processor Status Mode Enable(1) This bit enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. 1 = PSTAT mode enabled 0 = PSTAT mode disabled

1. PSTAT Mode is intended for factory processor debug only. The PSTAT_EN bit should be written to disable PSTAT mode if Nexus messaging is desired. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled.

Table 628. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
0	SYS_CLK(1)
1	SYS_CLK/2
2	SYS_CLK/3
3	SYS_CLK/4
4	Reserved
5	Reserved
6	Reserved
7	SYS_CLK/8

1. The SYS_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.

38.5 Functional description

38.5.1 NPC reset configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively.

Table 629 describes the NPC reset configuration options.

Table 629. NPC Reset Configuration Options

JCOMP Equal to npc_jcomp_plug?	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
no	X	X	Reset
yes	0	X	Disabled
yes	1	1	Full-Port Mode
yes	1	0	Reduced-Port Mode

38.5.2 Auxiliary Output Port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the Nexus modules and arbitrates for access to the port.

Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the MSEO functions. The MSEO pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and MSEO are sampled on the rising edge of MCKO.

Figure 668 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

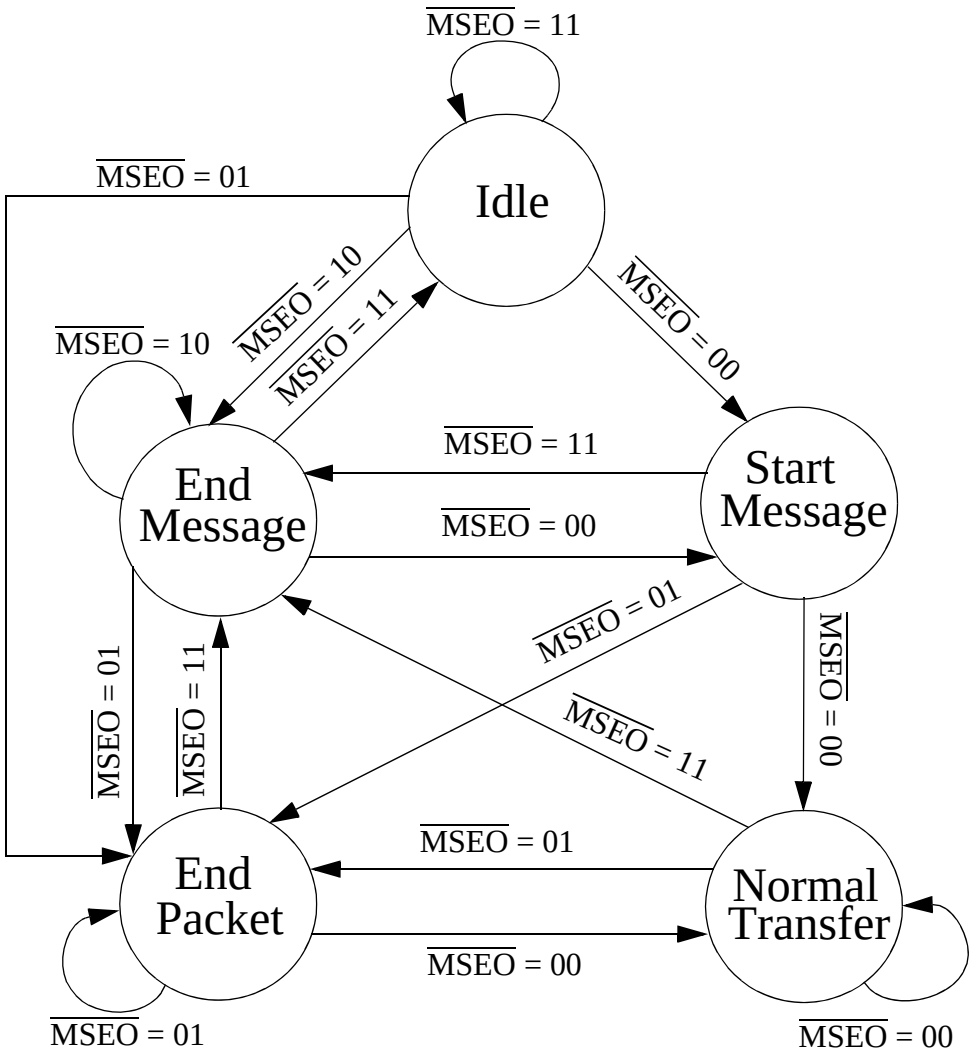


Figure 668. $\overline{\text{MSEO}}$ Transfers (for 2-bit $\overline{\text{MSEO}}$)

Output Messages

In addition to sending out messages generated in other Nexus blocks, the NPC can also output the device ID message contained in the device ID register and the port replacement output message on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 630 describes the device ID and port replacement output messages that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 630. NPC Output Messages

Message Name	Min. Packet Size (bits)	Max Packet Size (bits)	Packet Type	Packet Name	Packet Description
Device ID Message	6	6	fixed	TCODE	Value = 1
	32	32	fixed	ID	DID register contents

Figure 669 shows the various message formats that the pin interface formatter has to encounter. Note that for variable-length fields, the transmitted size of the field is determined from the range of the least significant bit to the most significant non-zero-valued bit (i.e. most significant zero-valued bits are not transmitted).

Figure 669. Message Field Sizes

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size[1] (bits)	Max Size[2] (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

NOTES:

1. Minimum information size. The actual number of bits transmitted depends on the number of MDO pins
2. Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

The double edges in *Figure 669* indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

Rules of Message

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field may start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets may start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. *Figure 670* shows the transmission sequence of a message that is made up of a TCODE followed by two fields.

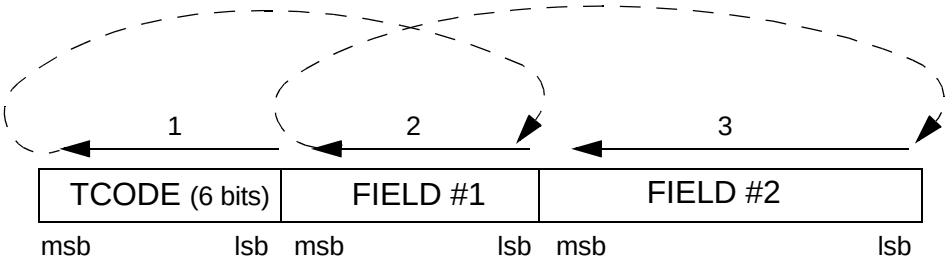


Figure 670. Transmission Sequence of Messages

38.5.3 IEEE 1149.1-2001 (JTAG) TAP

The NPC block uses the IEEE 1149.1-2001 TAP for accessing registers. Each of the individual Nexus blocks on the device implements a TAP controller for accessing its registers as well. TAP signals include TCK, TDI, TMS, and TDO. There may also be other blocks on the MCU that use the TAP and implement a TAP controller. The value of the JCOMP input controls ownership of the port between Nexus and non-Nexus blocks sharing the TAP.

Refer to the IEEE 1149.1-2001 specification for further detail on electrical and pin protocol compliance requirements.

The NPC implements a Nexus controller state machine that transitions based on the state of the IEEE 1149.1-2001 state machine shown in *Figure 672*. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2001 standard. It is shown in *Figure 673*.

The instructions implemented by the NPC TAP controller are listed in *Table 631*. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is 4-bits.

Table 631. Implemented Instructions

Instruction Name	Private/ Public	Opcode	Description
NEXUS-ENABLE	public	0x0	Activate Nexus controller state machine to read and write NPC registers.
BYPASS	private	0xF	NPC BYPASS instruction. Also the value loaded into the NPC IR upon exit of reset.

Data is shifted between TDI and TDO starting with the least significant bit as illustrated in *Figure 671*. This applies for the instruction register and all Nexus tool-mapped registers.

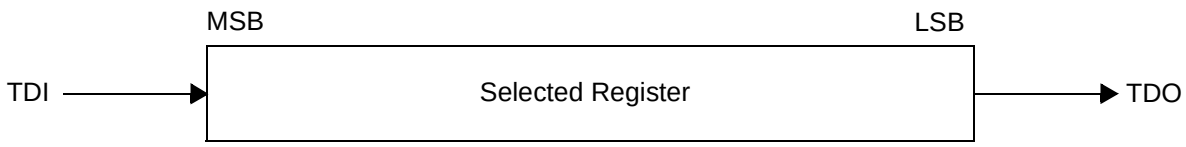
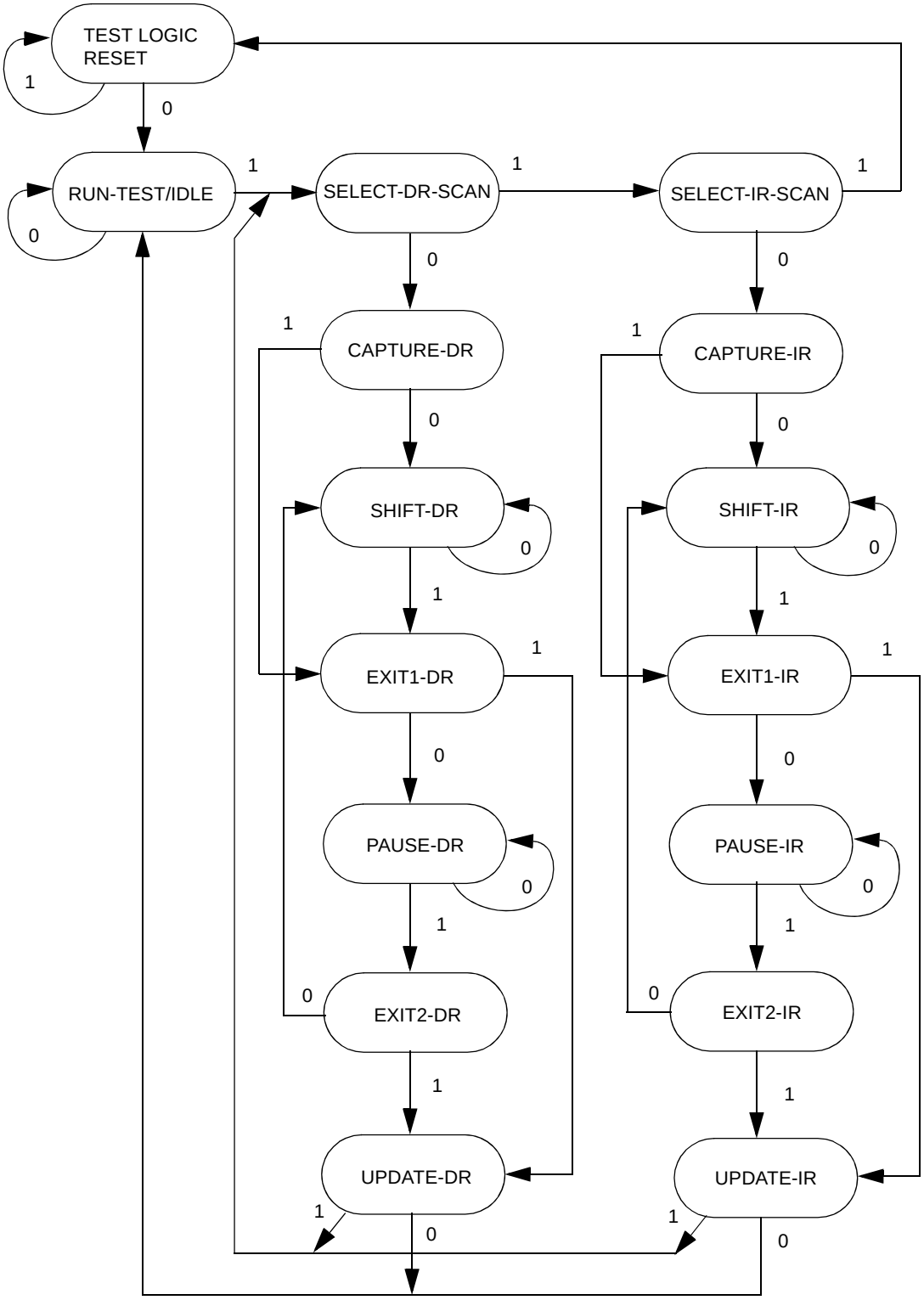


Figure 671. Shifting Data Into Register

Enabling the NPC TAP controller

Assertion of the power-on reset signal, entry into censored mode, or setting JCOMP to a value other than the NPC enable encoding resets the NPC TAP controller. When not in power-on reset or censored mode, the NPC TAP controller is enabled by driving JCOMP with the NPC enable value and exiting the Test-Logic-Reset state. Loading the NEXUS-ENABLE instruction then grants access to Nexus debug.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 672. IEEE 1149.1-2001 TAP Controller State Machine

Retrieving device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the Nexus Device ID register through the TAP. Transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR, if the NPC is enabled. Transmission of the device identification message serially via TDO is achieved by performing a read of the register contents as described in *Section , Selecting a Nexus Client Register*.

Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled when the TAP controller instruction register is loaded with the NEXUS-ENABLE instruction. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in *Figure 673*, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. *Table 632* illustrates the IEEE 1149.1 sequence to load the NEXUS-ENABLE instruction.

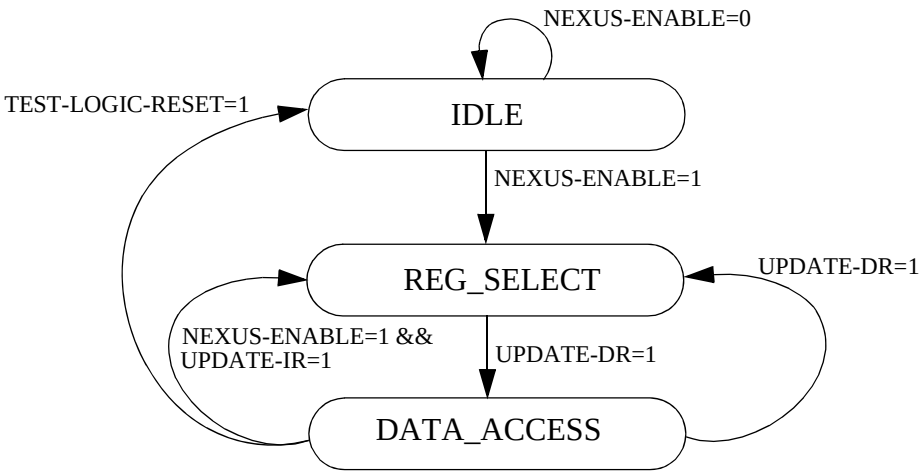


Figure 673. NEXUS Controller State Machine

Table 632. Loading NEXUS-ENABLE instruction

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	IDLE	Transitional state
2	1	SELECT-IR-SCAN	IDLE	Transitional state
3	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKS				
12	1	EXIT1-IR	IDLE	Last bit of instruction shifted in

Table 632. Loading NEXUS-ENABLE instruction (continued)

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
13	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
14	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

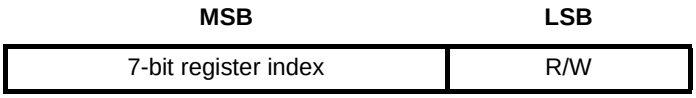
Selecting a Nexus Client Register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path. The Nexus Controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit register address index, as illustrated in *Figure 674*. The read/write control bit is set to 1 for writes and 0 for reads.

Figure 674. IEEE 1149.1 Controller Command Input



The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (LSB first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

Table 633 illustrates a sequence which writes a 32-bit value to a register

Table 633. Write to a 32-Bit Nexus Client Register

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
12	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
13	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
14	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path

Table 633. Write to a 32-Bit Nexus Client Register (continued)

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
15	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
16	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
48	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.
49	1	UPDATE-DR	DATA_ACCESS	Value written to register
50	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

38.5.4 Nexus JTAG Port Sharing

Each of the individual Nexus blocks on the device implements a TAP controller for accessing its registers. When Nexus has ownership of the TAP, only the block whose NEXUS-ENABLE instruction is loaded has control of the TAP. This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. If no register is selected as the shift path for a Nexus block, that block acts like a single-bit shift register, or bypass register.

38.5.5 MCKO and ipg_sync_mcko

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV field in the PCR. Possible operating frequencies include system clock, one-half system clock, one-quarter system clock, and one-eighth system clock speed.

The NPC also generates an MCKO clock gating control output signal. This output can be used by the MCKO generation logic to gate the transmission of MCKO when the auxiliary port is enabled but not transmitting messages. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is active. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1.

38.5.6 $\overline{\text{EVTO}}$ Sharing

The NPC block controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. The NPC assumes incoming $\overline{\text{EVTO}}$ signals will be asserted for one system clock period. After receiving a single clock period of asserted $\overline{\text{EVTO}}$ from any Nexus client, the NPC latches the result, and drives $\overline{\text{EVTO}}$ for one MCKO period on the following clock. When there is no active MCKO, such as in disabled mode, the NPC drives $\overline{\text{EVTO}}$ for two system clock periods. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

38.5.7 Nexus Reset Control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus blocks. If JCOMP is negated, an

internal reset is asserted, indicating that all Nexus modules should be held in reset. Internal Nexus reset is also asserted when the device is in censored mode.

38.5.8 System Clock Locked Indication

Following a power-on reset, MDO[0] can be monitored to provide the lock status of the system clock. MDO[0] is driven to a logic 1 until the system clock achieves lock after exiting power-on reset. Once the system clock is locked, MDO[0] is negated and tools may begin Nexus configuration. Loss of lock conditions that occur subsequent to the exit of power-on reset and the initial lock of the system clock do not cause a Nexus reset, and therefore do not result in MDO[0] driven high.

38.6 Initialization/Application Information

38.6.1 Accessing NPC tool-mapped registers

To initialize the TAP for Nexus register accesses, the following sequence is required:

1. Enable the Nexus TAP controller
2. Load the TAP controller with the NEXUS-ENABLE instruction

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE-ISTO 5001-2001 standard for more detail.

Appendix A Registers Under Protection

For SPC56xP60/54, the Register Protection module is operable on the registers listed in *Table 634*.

Table 634. Registers under protection

Module	Register	Register size (bits)	Register offset	Protected bitfields
Code Flash—Base address: 0xC3F8_8000 4 registers to protect				
Code Flash	MCR	32	0x0000	bits[0:31]
Code Flash	PFCR0	32	0x001C	bits[0:31]
Code Flash	PFCR1	32	0x0020	bits[0:31]
Code Flash	PFAPR	32	0x0024	bits[0:31]
Data Flash—Base address: 0xC3F8_C000 1 register to protect				
Data Flash	MCR	32	0x0000	bits[0:31]
SIU lite—Base address: 0xC3F9_0000 154 registers to protect				
SIUL	IRER	32	0x0018	bits[0:31]
SIUL	IREER	32	0x0028	bits[0:31]
SIUL	IFEER	32	0x002C	bits[0:31]
SIUL	IFER	32	0x0030	bits[0:31]
SIUL	PCR0	16	0x0040	bits[0:15]
SIUL	PCR1	16	0x0042	bits[0:15]
SIUL	PCR2	16	0x0044	bits[0:15]
SIUL	PCR3	16	0x0046	bits[0:15]
SIUL	PCR4	16	0x0048	bits[0:15]
SIUL	PCR5	16	0x004A	bits[0:15]
SIUL	PCR6	16	0x004C	bits[0:15]
SIUL	PCR7	16	0x004E	bits[0:15]
SIUL	PCR8	16	0x0050	bits[0:15]
SIUL	PCR9	16	0x0052	bits[0:15]
SIUL	PCR10	16	0x0054	bits[0:15]
SIUL	PCR11	16	0x0056	bits[0:15]
SIUL	PCR12	16	0x0058	bits[0:15]
SIUL	PCR13	16	0x005A	bits[0:15]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR14	16	0x005C	bits[0:15]
SIUL	PCR15	16	0x005E	bits[0:15]
SIUL	PCR16	16	0x0060	bits[0:15]
SIUL	PCR17	16	0x0062	bits[0:15]
SIUL	PCR18	16	0x0064	bits[0:15]
SIUL	PCR19	16	0x0066	bits[0:15]
SIUL	PCR20	16	0x0068	bits[0:15]
SIUL	PCR21	16	0x006A	bits[0:15]
SIUL	PCR22	16	0x006C	bits[0:15]
SIUL	PCR23	16	0x006E	bits[0:15]
SIUL	PCR24	16	0x0070	bits[0:15]
SIUL	PCR25	16	0x0072	bits[0:15]
SIUL	PCR26	16	0x0074	bits[0:15]
SIUL	PCR27	16	0x0076	bits[0:15]
SIUL	PCR28	16	0x0078	bits[0:15]
SIUL	PCR29	16	0x007A	bits[0:15]
SIUL	PCR30	16	0x007C	bits[0:15]
SIUL	PCR31	16	0x007E	bits[0:15]
SIUL	PCR32	16	0x0080	bits[0:15]
SIUL	PCR33	16	0x0082	bits[0:15]
SIUL	PCR48	16	0x00A0	bits[0:15]
SIUL	PCR49	16	0x00A2	bits[0:15]
SIUL	PCR50	16	0x00A4	bits[0:15]
SIUL	PCR51	16	0x00A6	bits[0:15]
SIUL	PCR52	16	0x00A8	bits[0:15]
SIUL	PCR53	16	0x00AA	bits[0:15]
SIUL	PCR54	16	0x00AC	bits[0:15]
SIUL	PCR55	16	0x00AE	bits[0:15]
SIUL	PCR56	16	0x00B0	bits[0:15]
SIUL	PCR57	16	0x00B2	bits[0:15]
SIUL	PCR58	16	0x00B4	bits[0:15]
SIUL	PCR59	16	0x00B6	bits[0:15]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR60	16	0x00B8	bits[0:15]
SIUL	PCR61	16	0x00BA	bits[0:15]
SIUL	PCR62	16	0x00BC	bits[0:15]
SIUL	PCR63	16	0x00BE	bits[0:15]
SIUL	PCR64	16	0x00C0	bits[0:15]
SIUL	PCR65	16	0x00C2	bits[0:15]
SIUL	PCR66	16	0x00C4	bits[0:15]
SIUL	PCR67	16	0x00C6	bits[0:15]
SIUL	PCR68	16	0x00C8	bits[0:15]
SIUL	PCR69	16	0x00CA	bits[0:15]
SIUL	PCR70	16	0x00CC	bits[0:15]
SIUL	PCR71	16	0x00CE	bits[0:15]
SIUL	PCR72	16	0x00D0	bits[0:15]
SIUL	PCR73	16	0x00D2	bits[0:15]
SIUL	PCR74	16	0x00D4	bits[0:15]
SIUL	PCR75	16	0x00D6	bits[0:15]
SIUL	PCR76	16	0x00D8	bits[0:15]
SIUL	PCR77	16	0x00DA	bits[0:15]
SIUL	PCR78	16	0x00DC	bits[0:15]
SIUL	PCR79	16	0x00DE	bits[0:15]
SIUL	PCR80	16	0x00E0	bits[0:15]
SIUL	PCR81	16	0x00E2	bits[0:15]
SIUL	PCR82	16	0x00E4	bits[0:15]
SIUL	PCR83	16	0x00E6	bits[0:15]
SIUL	PCR84	16	0x00E8	bits[0:15]
SIUL	PCR85	16	0x00EA	bits[0:15]
SIUL	PCR86	16	0x00EC	bits[0:15]
SIUL	PCR87	16	0x00EE	bits[0:15]
SIUL	PCR88	16	0x00F0	bits[0:15]
SIUL	PCR89	16	0x00F2	bits[0:15]
SIUL	PCR90	16	0x00F4	bits[0:15]
SIUL	PCR91	16	0x00F6	bits[0:15]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR92	16	0x00F8	bits[0:15]
SIUL	PCR93	16	0x00FA	bits[0:15]
SIUL	PCR94	16	0x00FC	bits[0:15]
SIUL	PCR95	16	0x00FE	bits[0:15]
SIUL	PCR96	16	0x0100	bits[0:15]
SIUL	PCR97	16	0x0102	bits[0:15]
SIUL	PCR98	16	0x0104	bits[0:15]
SIUL	PCR99	16	0x0106	bits[0:15]
SIUL	PCR100	16	0x0108	bits[0:15]
SIUL	PCR101	16	0x010A	bits[0:15]
SIUL	PCR102	16	0x010C	bits[0:15]
SIUL	PCR103	16	0x010E	bits[0:15]
SIUL	PCR104	16	0x0110	bits[0:15]
SIUL	PCR105	16	0x0112	bits[0:15]
SIUL	PCR106	16	0x0114	bits[0:15]
SIUL	PCR107	16	0x0116	bits[0:15]
SIUL	PSMI0_3	32	0x0500	bits[0:31]
SIUL	PSMI4_7	32	0x0504	bits[0:31]
SIUL	PSMI8_11	32	0x0508	bits[0:31]
SIUL	PSMI12_15	32	0x050C	bits[0:31]
SIUL	PSMI28_31	32	0x051C	bits[0:31]
SIUL	PSMI32_35	32	0x0520	bits[0:31]
SIUL	PSMI36_39	32	0x0524	bits[0:31]
SIUL	IFMC0	32	0x1000	bits[0:31]
SIUL	IFMC1	32	0x01004	bits[0:31]
SIUL	IFMC2	32	0x1008	bits[0:31]
SIUL	IFMC3	32	0x100C	bits[0:31]
SIUL	IFMC4	32	0x1010	bits[0:31]
SIUL	IFMC5	32	0x1014	bits[0:31]
SIUL	IFMC6	32	0x1018	bits[0:31]
SIUL	IFMC7	32	0x101C	bits[0:31]
SIUL	IFMC8	32	0x1020	bits[0:31]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	IFMC9	32	0x1024	bits[0:31]
SIUL	IFMC10	32	0x1028	bits[0:31]
SIUL	IFMC11	32	0x102C	bits[0:31]
SIUL	IFMC12	32	0x1030	bits[0:31]
SIUL	IFMC13	32	0x1034	bits[0:31]
SIUL	IFMC14	32	0x1038	bits[0:31]
SIUL	IFMC15	32	0x103C	bits[0:31]
SIUL	IFMC16	32	0x1040	bits[0:31]
SIUL	IFMC17	32	0x1044	bits[0:31]
SIUL	IFMC18	32	0x1048	bits[0:31]
SIUL	IFMC19	32	0x104C	bits[0:31]
SIUL	IFMC20	32	0x1050	bits[0:31]
SIUL	IFMC21	32	0x1054	bits[0:31]
SIUL	IFMC22	32	0x1058	bits[0:31]
SIUL	IFMC23	32	0x105C	bits[0:31]
SIUL	IFMC24	32	0x1060	bits[0:31]
SIUL	IFMC25	32	0x1064	bits[0:31]
SIUL	IFMC26	32	0x1068	bits[0:31]
SIUL	IFMC27	32	0x106C	bits[0:31]
SIUL	IFMC28	32	0x1070	bits[0:31]
SIUL	IFMC29	32	0x1074	bits[0:31]
SIUL	IFMC30	32	0x1078	bits[0:31]
SIUL	IFMC31	32	0x107C	bits[0:31]
SIUL	IFCP	32	0x1080	bits[0:31]
Power Management Unit—Base address: 0xC3FE_8080 1 register to protect				
PMU	VREG_CTL	32	0x0000	bits[0:31]
MC Mode Entry—Base address: 0xC3FD_C000 36 registers to protect				
MC ME	ME_ME	32	0x0008	bits[0:31]
MC ME	ME_IM	32	0x0010	bits[0:31]
MC ME	ME_TEST_MC	32	0x0024	bits[0:31]
MC ME	ME_SAFE_MC	32	0x0028	bits[0:31]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
MC ME	ME_DRUN_MC	32	0x002C	bits[0:31]
MC ME	ME_RUN0_MC	32	0x0030	bits[0:31]
MC ME	ME_RUN1_MC	32	0x0034	bits[0:31]
MC ME	ME_RUN2_MC	32	0x0038	bits[0:31]
MC ME	ME_RUN3_MC	32	0x003C	bits[0:31]
MC ME	ME_HALT0_MC	32	0x0040	bits[0:31]
MC ME	ME_STOP0_MC	32	0x0048	bits[0:31]
MC ME	ME_RUN_PC0	32	0x0080	bits[0:31]
MC ME	ME_RUN_PC1	32	0x0084	bits[0:31]
MC ME	ME_RUN_PC2	32	0x0088	bits[0:31]
MC ME	ME_RUN_PC3	32	0x008C	bits[0:31]
MC ME	ME_RUN_PC4	32	0x0090	bits[0:31]
MC ME	ME_RUN_PC5	32	0x0094	bits[0:31]
MC ME	ME_RUN_PC6	32	0x0098	bits[0:31]
MC ME	ME_RUN_PC7	32	0x009C	bits[0:31]
MC ME	ME_LP_PC0	32	0x00A0	bits[0:31]
MC ME	ME_LP_PC1	32	0x00A4	bits[0:31]
MC ME	ME_LP_PC2	32	0x00A8	bits[0:31]
MC ME	ME_LP_PC3	32	0x00AC	bits[0:31]
MC ME	ME_LP_PC4	32	0x00B0	bits[0:31]
MC ME	ME_LP_PC5	32	0x00B4	bits[0:31]
MC ME	ME_LP_PC6	32	0x00B8	bits[0:31]
MC ME	ME_LP_PC7	32	0x00BC	bits[0:31]
MC ME	ME_PCTL[4..7]	32	0x00C4	bits[0:31]
MC ME	ME_PCTL[16..19]	32	0x00D0	bits[0:31]
MC ME	ME_PCTL[24..27]	32	0x00D8	bits[0:31]
MC ME	ME_PCTL[32..35]	32	0x00E0	bits[0:31]
MC ME	ME_PCTL[36..39]	32	0x00E4	bits[0:31]
MC ME	ME_PCTL[40..43]	32	0x00E8	bits[0:31]
MC ME	ME_PCTL[48..51]	32	0x00F0	bits[0:31]
MC ME	ME_PCTL[84.87]	32	0x0114	bits[0:31]
MC ME	ME_PCTL[92..95]	32	0x011C	bits[0:31]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
MC Clock Generation Module—Base address: 0xC3FE_0000 8 registers to protect				
MC CGM	CGM_OC_EN	8	0x0370	bits[0:7]
MC CGM	CGM_OCDS_SC	8	0x0374	bits[0:7]
MC CGM	CGM_AC0_SC	8	0x0380	bits[0:31]
MC CGM	CGM_AC0_DC[0..3]	8	0x0384	bits[0:31]
MC CGM	CGM_AC1_SC	8	0x0388	bits[0:31]
MC CGM	CGM_AC1_DC[0..3]	8	0x038C	bits[0:31]
MC CGM	CGM_AC2_SC	8	0x0390	bits[0:31]
MC CGM	CGM_AC2_DC[0..3]	8	0x0394	bits[0:31]
XOSC—Base address: 0xC3FE_0000 1 register to protect				
XOSC	OSC_CTL	32	0x0000	bits[0:31]
IRC_OSC—Base address: 0xC3FE_0060 1 register to protect				
IRC_OSC	RC_CTL	32	0x0000	bits[0:31]
FM PLL 0—Base address: 0xC3FE_00A0 2 registers to protect				
FMPLL 0	CR	32	0x0000	bits[0:31]
FMPLL 0	MR	32	0x0004	bits[0:31]
CMU 0—Base address: 0xC3FE_0100 1 register to protect				
CMU 0	CMU_CSR	32	0x0000	bits[0:31]
CMU 1—Base address: 0xC3FE_0120 1 register to protect				
CMU 1	CMU_CSR	32	0x0000	bits[0:31]
MC Reset Generation Module—Base address: 0xC3FE_4000 6 registers to protect				
MC RGM	RGM_FERD	16	0x0004	bits[0:15]
MC RGM	RGM_DERD	16	0x0006	bits[0:15]
MC RGM	RGM_FEAR	16	0x0010	bits[0:15]
MC RGM	RGM_DEAR	16	0x0012	bits[0:15]
MC RGM	RGM_FESS	16	0x0018	bits[0:15]
MC RGM	RGM_FBRE	16	0x001C	bits[0:15]

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
PIT_RTI—Base address: 0xC3FF_0000 9 registers to protect				
PIT_RTI	PIT_RTI_PITMCR	32	0x0000	32-bit
PIT_RTI	PIT_RTI_LDVAL0	32	0x0100	32-bit
PIT_RTI	PIT_RTI_TCTRL0	32	0x0108	32-bit
PIT_RTI	PIT_RTI_LDVAL1	32	0x0110	32-bit
PIT_RTI	PIT_RTI_TCTRL1	32	0x0118	32-bit
PIT_RTI	PIT_RTI_LDVAL2	32	0x0120	32-bit
PIT_RTI	PIT_RTI_TCTRL2	32	0x0128	32-bit
PIT_RTI	PIT_RTI_LDVAL3	32	0x0130	32-bit
PIT_RTI	PIT_RTI_TCTRL3	32	0x0138	32-bit
ADC 0—Base address: 0xFFE0_0000 10 registers to protect				
ADC 0	CLR0	32	0x0000	32-bit
ADC 0	CLR1	32	0x0004	32-bit
ADC 0	CLR2	32	0x0008	32-bit
ADC 0	CLR3	32	0x000C	32-bit
ADC 0	CLR4	32	0x0010	32-bit
ADC 0	TRC0	32	0x0034	32-bit
ADC 0	TRC1	32	0x0038	32-bit
ADC 0	TRC2	32	0x003C	32-bit
ADC 0	TRC3	32	0x0040	32-bit
ADC 0	PREREG	32	0x00A8	32-bit
eTimer 0—Base address: 0xFFE1_8000 24 registers to protect				
eTimer 0	CH0_CTRL	16	0x000E	16-bit
eTimer 0	CH0_CTRL2	16	0x0010	16-bit
eTimer 0	CH0_CTRL3	16	0x0012	16-bit
eTimer 0	CH0_CCCTRL	16	0x001C	16-bit
eTimer 0	CH1_CTRL	16	0x002E	16-bit
eTimer 0	CH1_CTRL2	16	0x0030	16-bit
eTimer 0	CH1_CTRL3	16	0x0032	16-bit
eTimer 0	CH1_CCCTRL	16	0x003C	16-bit

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
eTimer 0	CH2_CTRL	16	0x004E	16-bit
eTimer 0	CH2_CTRL2	16	0x0050	16-bit
eTimer 0	CH2_CTRL3	16	0x0052	16-bit
eTimer 0	CH2_CCCTRL	16	0x005C	16-bit
eTimer 0	CH3_CTRL	16	0x006E	16-bit
eTimer 0	CH3_CTRL2	16	0x0070	16-bit
eTimer 0	CH3_CTRL3	16	0x0072	16-bit
eTimer 0	CH3_CCCTRL	16	0x007C	16-bit
eTimer 0	CH4_CTRL	16	0x008E	16-bit
eTimer 0	CH4_CTRL2	16	0x0090	16-bit
eTimer 0	CH4_CTRL3	16	0x0092	16-bit
eTimer 0	CH4_CCCTRL	16	0x009C	16-bit
eTimer 0	CH5_CTRL	16	0x00AE	16-bit
eTimer 0	CH5_CTRL2	16	0x00B0	16-bit
eTimer 0	CH5_CTRL3	16	0x00B2	16-bit
eTimer 0	CH5_CCCTRL	16	0x00BC	16-bit
eTimer 1—Base address: 0xFFE1_C000 24 registers to protect				
eTimer 1	CH0_CTRL	16	0x000E	16-bit
eTimer 1	CH0_CTRL2	16	0x0010	16-bit
eTimer 1	CH0_CTRL3	16	0x0012	16-bit
eTimer 1	CH0_CCCTRL	16	0x001C	16-bit
eTimer 1	CH1_CTRL	16	0x002E	16-bit
eTimer 1	CH1_CTRL2	16	0x0030	16-bit
eTimer 1	CH1_CTRL3	16	0x0032	16-bit
eTimer 1	CH1_CCCTRL	16	0x003C	16-bit
eTimer 1	CH2_CTRL	16	0x004E	16-bit
eTimer 1	CH2_CTRL2	16	0x0050	16-bit
eTimer 1	CH2_CTRL3	16	0x0052	16-bit
eTimer 1	CH2_CCCTRL	16	0x005C	16-bit
eTimer 1	CH3_CTRL	16	0x00x006E	16-bit
eTimer 1	CH3_CTRL2	16	0x0070	16-bit
eTimer 1	CH3_CTRL3	16	0x0072	16-bit

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
eTimer 1	CH3_CCCTRL	16	0x007C	16-bit
eTimer 1	CH4_CTRL	16	0x008E	16-bit
eTimer 1	CH4_CTRL2	16	0x0090	16-bit
eTimer 1	CH4_CTRL3	16	0x0092	16-bit
eTimer 1	CH4_CCCTRL	16	0x009C	16-bit
eTimer 1	CH5_CTRL	16	0x00AE	16-bit
eTimer 1	CH5_CTRL2	16	0x00B0	16-bit
eTimer 1	CH5_CTRL3	16	0x00B2	16-bit
eTimer 1	CH5_CCCTRL	16	0x00BC	16-bit
CRC_0—Base address: 0xFFE6_8000 3 registers to protect				
CRC_0	CRC0_CFG0	32	0x0000	16-bit
CRC_0	CRC0_CFG1	32	0x0010	16-bit
CRC_0	CRC0_CFG2	32	0x0020	16-bit
CRC_1—Base address: 0x9FE7_0000 3 registers to protect				
CRC_1	CRC1_CFG0	32	0x0000	16-bit
CRC_1	CRC1_CFG1	32	0x0010	16-bit
CRC_1	CRC1_CFG2	32	0x0020	16-bit
DSPI 0—Base address: 0xFFF9_0000 11 registers to protect				
DSPI 0	DSPI_MCR	32	0x0000	32-bit
DSPI 0	DSPI_TCR	32	0x0008	32-bit
DSPI 0	DSPI_CTAR0	32	0x000C	32-bit
DSPI 0	DSPI_CTAR1	32	0x0010	32-bit
DSPI 0	DSPI_CTAR2	32	0x0014	32-bit
DSPI 0	DSPI_CTAR3	32	0x0018	32-bit
DSPI 0	DSPI_CTAR4	32	0x001C	32-bit
DSPI 0	DSPI_CTAR5	32	0x0020	32-bit
DSPI 0	DSPI_CTAR6	32	0x0024	32-bit
DSPI 0	DSPI_CTAR7	32	0x0028	32-bit
DSPI 0	DSPI_RSER	32	0x0030	32-bit

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
DSPI 1—Base address: 0xFFF9_4000 11 registers to protect				
DSPI 1	DSPI_MCR	32	0x0000	32-bit
DSPI 1	DSPI_TCR	32	0x0008	32-bit
DSPI 1	DSPI_CTAR0	32	0x000C	32-bit
DSPI 1	DSPI_CTAR1	32	0x0010	32-bit
DSPI 1	DSPI_CTAR2	32	0x0014	32-bit
DSPI 1	DSPI_CTAR3	32	0x0018	32-bit
DSPI 1	DSPI_CTAR4	32	0x001C	32-bit
DSPI 1	DSPI_CTAR5	32	0x0020	32-bit
DSPI 1	DSPI_CTAR6	32	0x0024	32-bit
DSPI 1	DSPI_CTAR7	32	0x0028	32-bit
DSPI 1	DSPI_RSER	32	0x0030	32-bit
DSPI 2—Base address: 0xFFF9_8000 11 registers to protect				
DSPI 2	DSPI_MCR	32	0x0000	32-bit
DSPI 2	DSPI_TCR	32	0x0008	32-bit
DSPI 2	DSPI_CTAR0	32	0x000C	32-bit
DSPI 2	DSPI_CTAR1	32	0x0010	32-bit
DSPI 2	DSPI_CTAR2	32	0x0014	32-bit
DSPI 2	DSPI_CTAR3	32	0x0018	32-bit
DSPI 2	DSPI_CTAR4	32	0x001C	32-bit
DSPI 2	DSPI_CTAR5	32	0x0020	32-bit
DSPI 2	DSPI_CTAR6	32	0x0024	32-bit
DSPI 2	DSPI_CTAR7	32	0x0028	32-bit
DSPI 2	DSPI_RSER	32	0x0030	32-bit
DSPI 3—Base address: 0xFFF9_C000 11 registers to protect				
DSPI 3	DSPI_MCR	32	0x0000	32-bit
DSPI 3	DSPI_TCR	32	0x0008	32-bit
DSPI 3	DSPI_CTAR0	32	0x000C	32-bit
DSPI 3	DSPI_CTAR1	32	0x0010	32-bit
DSPI 3	DSPI_CTAR2	32	0x0014	32-bit

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
DSPI 3	DSPI_CTAR3	32	0x0018	32-bit
DSPI 3	DSPI_CTAR4	32	0x001C	32-bit
DSPI 3	DSPI_CTAR5	32	0x0020	32-bit
DSPI 3	DSPI_CTAR6	32	0x0024	32-bit
DSPI 3	DSPI_CTAR7	32	0x0028	32-bit
DSPI 3	DSPI_RSER	32	0x0030	32-bit
DSPI 4—Base address: 0x8FFA_C000 11 registers to protect				
DSPI 4	DSPI_MCR	32	0x0000	32-bit
DSPI 4	DSPI_TCR	32	0x0008	32-bit
DSPI 4	DSPI_CTAR0	32	0x000C	32-bit
DSPI 4	DSPI_CTAR1	32	0x0010	32-bit
DSPI 4	DSPI_CTAR2	32	0x0014	32-bit
DSPI 4	DSPI_CTAR3	32	0x0018	32-bit
DSPI 4	DSPI_CTAR4	32	0x001C	32-bit
DSPI 4	DSPI_CTAR5	32	0x0020	32-bit
DSPI 4	DSPI_CTAR6	32	0x0024	32-bit
DSPI 4	DSPI_CTAR7	32	0x0028	32-bit
DSPI 4	DSPI_RSER	32	0x0030	32-bit
FlexCAN_0—Base address: 0xFFFC_0000 7 registers to protect				
FlexCAN_0	CANx_MCR	32	0x0000	32-bit
FlexCAN_0	CANx_CTRL	32	0x0004	32-bit
FlexCAN_0	CANx_RXGMASK	32	0x0010	32-bit
FlexCAN_0	CANx_RX14MASK	32	0x0014	32-bit
FlexCAN_0	CANx_RX15MASK	32	0x0018	32-bit
FlexCAN_0	CANx_IMASK2	32	0x0024	32-bit
FlexCAN_0	CANx_IMASK	32	0x0028	32-bit
FlexCAN_1—Base address: 0x8FFC_4000 7 registers to protect				
FlexCAN_1	CANx_MCR	32	0x0000	32-bit
FlexCAN_1	CANx_CTRL	32	0x0004	32-bit
FlexCAN_1	CANx_RXGMASK	32	0x0010	32-bit

Table 634. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
FlexCAN_1	CANx_RX14MASK	32	0x0014	32-bit
FlexCAN_1	CANx_RX15MASK	32	0x0018	32-bit
FlexCAN_1	CANx_IMASK2	32	0x0024	32-bit
FlexCAN_1	CANx_IMASK	32	0x0028	32-bit
FlexRay—Base address: 0xFFFE_0000 1 register to protect				
FlexRay	MCR	16	0x0002	16-bit
Safety Port—Base address: 0xFFFE_8000 7 registers to protect				
Safety port	CANx_MCR	32	0x0000	32-bit
Safety port	CANx_CTRL	32	0x0004	32-bit
Safety port	CANx_RXGMASK	32	0x0010	32-bit
Safety port	CANx_RX14MASK	32	0x0014	32-bit
Safety port	CANx_RX15MASK	32	0x0018	32-bit
Safety port	CANx_IMASK2	32	0x0024	32-bit
Safety port	CANx_IMASK	32	0x0028	32-bit

Revision History

Table 635 summarizes revisions to this document.

Table 635. Revision history

Date	Revision	Substantive Changes
08-Apr-2011	1	Initial release
29-Jun-2011	2	Added “Memory Protection Unit (MPU)” Chapter. In the “Memory map” section: – Changed the offset value of “Data Flash array 0: test sector“, was 0x00C0_0000, now is 0x00C0_2000. In the “Signal Description” chapter: – In the “CTU / ADCs / eTimers / DSPI / FlexRay connections” table: Replaced all occurences of “new logic IP” with “CTU/ADC IP Interface” Added a footnote. In the “Clock Description” chapter: – Rewrote the sections: “Auxiliary Clock Selector 3 for FR_PLL divider (FlexRay clock)” and “Auxiliary Clock Dividers”. – In the “SPC56xP60/54 system clock generation” figure, replaced the content of “Clock Out Divider 256” with “÷ 1, ÷ 2, ÷ 3 to ÷ 256”. – In the “SPC56xP60/54 system clock generation” figure, removed the line between IRC_PLL and SYS_CLK. – Updated “Clock selectors” subsections. – In the “Clock architecture” section, added a note concerning the software compatibility. – Added “Output Clock Division 256 Select Register (CLK_DIV_256)” section. – Added “Core_1 switching off” section. In the “Clock Generation Module (MC_CGM)” chapter: – In the “MC_CGM Memory Map” table: added CMU1 register added CLK_DIV256 register In the Interrupt Controller (INTC)” chapter: – Removed Platform Flash Bank (2,3) abort and Platform Flash Bank (2,3) stall from Interrupt vector table. – Replaced “INTC block diagram” figure. – Replaced “ Interrut Vector” table. In the “System Status and Configuration Module (SSCM)” chapter: – Removed DMA status from feature list. – Added DPMBOOT, DPMKEY, UOPS, and PSA registers. – In the STATUS register, added CER, NXEN1, and VLE field descriptions.

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
29-Jun-2011	2 cont'd	In the "System Integration Unit Lite (SIUL)" chapter: – Replaced PSMI[0_3:32_35] with PSMI[0_3:36_39] through the chapter. – Updated "Pad selection" table. – In the "System Integration Unit Lite block diagram" figure, replaced 106 wth 108. In the "e200z0 and e200z0h Core" chapter: – Added "e200z0h Supervisor mode programmer's model" figure. Replaced "Peripheral Bridge (PBRIDGE)" chapter. Replaced "Crossbar Switch (XBAR)" chapter. In the "Error Correction Status Module (ECSM)" chapter: – In the "Introduction" section, removed " It also provides with register access protection for the following slave modules: INTC, ECSM, STM, and SWT" sentence. – In the "Features" section, removed "Capability to restrict register access to supervisor mode to selected on-platform slave devices: INTC, ECSM, STM, and SWT" bullet. – Updated MUDCR register, Removed MUDCR[31] and added MUDCR[30] – Updated contets of "RAM syndrome mapping for single-bit correctable errors" table. – Removed "ECSM_reg_protection" section. In the "Flash Memory" chapter: – Updated "SPC56xP60/54 Flash memory architecture" figure. – In the "Code Flash module structure" figure, added "Filter Cap" block. In the "LIN Controller (LINFlex)" chapter: – In the "Introduction" and "Main features" sections, removed LIN protocol versions 2.1 and J2602 because not supported. In the "FlexCAN" chapter: – In the "FlexCAN block diagram" figure: replaced "64/128/256-byte RAM" with "128-byte RAM" replaced "288/544/1056-byte RAM" with "544-byte RAM" – In the "FlexCAN memory mapping" section, removed reference to 16 MB condiguration. – In the "FlexCAN module memory map" table, "0x8FFC_0000 (CAN_1)" with "0x8FFC_4000 (CAN_1)". – In the "Module Configuration Register (MCR)" section: Removed WAK_MSW field. – In the SOTF_RST field descritpion, replaced RXIMR0–RXIMR63 with RXIMR0–RXIMR32. – In the "Error and Status Register (ESR)" register, removed WAK_INT bit field. – In the "Interrupts" section, addd a footnote about wakeup interrupt. – In the "FlexCAN initialization sequence" section, removed paragraphs concerning SOFT_RST.

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
29-Jun-2011	2 cont'd	In the "Analog-to-Digital Converter (ADC)" chapter: – Update the chapter in order to have 27 internal precision cannels. In the "Cross Triggering Unit (CTU)" chapter: – In the "Introduction" section, removed the first paragraph. – Replaced "ADC commands translation" table. – In the "ADC results" section, for FIFO1 and FIFO2 there are only 4 entries instead of 16. In the "Semaphore Unit (SEMA4)": – in the "introduction" section, removed the paragraph regarding dual processor mode and Lock Step mode because SPC56xP60/54 operates in Decoupled Parallel Mode(DPM) only. Replaced "Fault Collection and Control Unit (FCCU)" chapter. In the "Cyclic Redundancy Check (CRC)": – In the "CRC top level pinout" table, added "crc_compare_error" row. – Added CRC_OUTP_CHK register. – In the CRC_CFG register: added LEN bit fiked made POLYG bit fiked 2 bits long – In the "DMA-CRC Transmission Sequence" figure, replaced CPU with DMA in the phase 3 scheme. – Updated "DMA-CRC Reception Sequence" figure. In the "Voltage Regulators and Power Supplies" chapter: – Replaced "Five VDD_LV/VSS_LV pins pairs" with "Four VDD_LV/VSS_LV pins pairs". In the "IEEE 1149.1 Test Access Port Controller (JTAGC)" chapter: – In the "JTAG instruction" table, added: ACCESS_AUX_TAP_CORE0, ACCESS_AUX_TAP_CORE1, ACCESS_AUX_TAP_NASPS_0, and ACCESS_AUX_TAP_NASPS_1 rows. – In the "TAP sharing mode" section replaced ACCESS_AUX_TAP_ONCE with "ACCESS_AUX_TAP_CORE0, ACCESS_AUX_TAP_CORE1, ACCESS_AUX_TAP_NASPS_0, ACCESS_AUX_TAP_NASPS_1".

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
06-Jun-2012	3	*Chapter 1: Introduction* *Section 1.7.5, On-chip SRAM with ECC*: replaced two occurrences of “3 wait states” to “2 wait states” replaced 60 MHz to 64 MHz *Chapter 2: Memory Map* *Table 4: Memory map*: changed the “End address” and the “Size” values of SRAM added footonote about “On-chip Peripherals” changed the start address from “0x8FF7_4000” to “0x9FE7_4000” *Chapter 4: Clock Description* Added registers for CMU_1 Replaced all occurrences of “FMPLL_1” with “SYS_CLK” *Figure 7: SPC56xP60/54 system clock generation*, changed FMPLL_1D1_CLK to FMPLL_0_D1_CLK and added “FMPLL_0_D1_CLK— 80 MHz, 50%” Updated *Section : PLL clock monitor* Updated *Section : System clock monitor* *Table 13: OSC_CTL memory map*,removed access and reset columns per new agreement *Table 15: FMPLL memory map*,removed access and reset columns per new agreement *Table 20: CMU memory map*,removed access and reset columns per new agreement. *Figure 9: SPC56xP60/54 system clock distribution Part B*, replaced “ Magic Carpet” with proper content *Figure 18: SPC56xP60/54 with two CMUs*, replaced “ Magic Carpet” with proper content. *Section 4.7: IRC 16 MHz internal RC oscillator (RC_CTL)*, reworded register description *Section : High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_0)*, replaced CMU_0_HFREFR_A with CMU_0_HFREFR_0 *Section : Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_0)*, replaced CMU_0_LFREFR_A with CMU_0_LFREFR_0. *Chapter 6: Mode Entry Module (MC_ME)* *Table 53: Debug Mode Transition Status Register (ME_DMTS) Field Descriptions*, added description of SCSRC_SC bit field *Chapter 8: Reset Generation Module (MC_RGM)* Replaced all occurrences of RESET_B to $\overline{\text{RESET}}$ *Table 66: MC_RGM Memory Map*, removed D_COMP entry. *Table 70: Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions*, removed D_COMP entry. *Chapter 7: Power Control Unit (MC_PCU)* Added PCU_PCONF0, VREG_VCTL and VREG_STATUS *Chapter 9: Interrupt Controller (INTC)* Through whole chapter, renamed “INTC_PSR260” (was NTC_PSR258_260) *Chapter 10: System Status and Configuration Module (SSCM)* *Table 99: DPMKEY Field Descriptions*, added a footnote about reset value of VLE bit field.

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
06-Jun-2012	3 cont'd	*Chapter 11: System Integration Unit Lite (SIUL)* *Table 104: MIDR1 field descriptions*, in the PARTNUM description: replaced "0101_0110_0000_0100: 512 KB" with "0101_0110_0000_0101: 768 KB" added single/dual core details Added *Table 113: PCR bit implementation by pad type* *Figure 122 (Parallel GPIO Pad Data Out register 0–3(PGPDO[0:3]))*, changed offset of PGPDO2 to 0x0C08 *Table 104: MIDR1 field descriptions*, in the PARTNUM description, added single/dual core details *Chapter 13: Peripheral Bridge (PBRIDGE)* *Table 123: PBRIDGE registers*,removed access and reset columns per new agreement *Table 124: PBRIDGE memory map*, added reserved area from 0x0030 to 0x003F *Chapter 14: Crossbar Switch (XBAR)* *Table 133: XBAR register configuration summary*, changed column description (was "XBAR base offset", is "Offset from XBAR_BASE (0xFFF0_4000)") *Chapter 16: Internal Static RAM (SRAM)* *Table 162: SRAM memory map*, added a column *Chapter 17: Flash Memory* *Figure 173: Data Flash module structure*, changed "Test Sector" size from 16K to 8K *Table 164: Flash-related regions in the system memory map*, changed "Data Flash Array 0 Test Sector" size from 16K to 8K *Table 171: TestFlash structure*, changed reserved area for "Code TestFlash" and "Data TestFlash" *Table 173: Flash registers*, added note for UT2, UMISR2, UMISR3 and UMISR4 registers *Table 175: Flash 64 KB bank1 register map*, replaced UT2, UMISR2, UMISR3 and UMISR4 registers with reserved space *Section 17.3.2: Main features*, removed "One Time Programmable (OTP) area in TestFlash block" *Figure 185 (Platform Flash Configuration Register 0 (PFCR0))* updated reset value *Table 185: PFCR0 field descriptions*: updated bit filed name of bit 15, 16, and 24, respectively BK0_RWWC2, BK0_RWWC1, and BK0_RWWC0 added B0_P1_BCFG, B0_P1_DPFE, B0_P1_IPFE, B0_P1_PFLM, and B0_P1_BFE bit fields updated reset value *Table 186: PFCR1 field descriptions*: updated bit filed name of bit 15, 16, and 24, respectively BK1_RWWC2, BK1_RWWC1, and BK1_RWWC0 updated reset value *Table 202: NVUSRO field descriptions*: in the WATCHDOG_EN description, added a note "The WATCHDOG_EN bit is available only for SWT_0" removed OSCILLATOR_MARGIN field and updated UOx bit fields *Figure 203: Non-Volatile User Options register (NVUSRO)*: removed OSCILLATOR_MARGIN bit

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
06-Jun-2012	3 cont'd	Added note in *Section : User Test 2 register (UT2)*, *Section : User Multiple Input Signature Register 2 (UMISR2)*, *Section : User Multiple Input Signature Register 3 (UMISR3)* and *Section : User Multiple Input Signature Register 4 (UMISR4)* Renamed *Section 17.3.8: Code Flash programming considerations* (was "Programming considerations") *Section 17.2.4: Memory map and registers description*, added a note *Chapter 19: Enhanced Direct Memory Access (eDMA)* Through whole chapter, replaced all occurrences of "EDMA_CERR" with "EDMA_CER" Renamed *Figure 218: eDMA Enable Request Register (EDMA_ERQRL)* (was "eDMA Enable Request Low Register (EDMA_ERQRL)") Renamed *Figure 219: eDMA Enable Error Interrupt Register (EDMA_EEIRL)* (was "eDMA Enable Error Interrupt Low Register (EDMA_EEIRL)") *Chapter 23: LIN Controller (LINFlex)* *Figure 424: LIN status register (LINSR)*, changed LINS access from read/write to write only *Figure 429: LIN output compare register (LINOCR)*, changed note from LINTCSR[LTOM] = 1 to LINTCSR[LTOM] = 0 Added *Section : Overrun* *Section : Data transmission (transceiver as publisher)*, changed BDAR register with BDR register *Section : Filter mode*, changed sentence "eight IFCR registers" with "sixteen IFCR registers" *Section : Identifier filter mode configuration*, changed sentence "the filter must first be deactivated by programming IFER[FACT] = 0" with "the filter must first be activated by programming IFER[FACT] = 1" *Section : Automatic resynchronization method* now is a section *Section : LIN timeout mode*, changed sentence "Setting the LTOM bit" with "Resetting the LTOM bit" *Section : Output compare mode*, changed sentence "Programming LINTCSR[LTOM] = 0 enables the output compare mode" with "Programming LINTCSR[LTOM] = 1 enables the output compare mode" *Section : Identifier filter enable register (IFER)*: *Figure 438: Identifier filter enable register (IFER)*: changed FACT field from 8 bit to 16 bit Updated *Table 415: IFER field descriptions* Removed "IFER[FACT] configuration" table *Section : Identifier filter control register (IFCR2n)*: replaced note "This register can be written in Initialization mode only." with "Register bit can be read in any mode, written only in Initialization mode" *Figure 441: Identifier filter control register (IFCR2n)*, changed ID access from read/write "w1c" to read/write only in initialization mode *Section : Identifier filter control register (IFCR2n + 1)*: replaced note "This register can be written in Initialization mode only." with "Register bit can be read in any mode, written only in Initialization mode" *Figure 442: Identifier filter control register (IFCR2n + 1)*, changed ID access from read/write "w1c" to read/write only in initialization mode *Chapter 24: FlexCAN* *Section 24.3.3: Rx FIFO structure*, update address values through the section.

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
06-Jun-2012	3 cont'd	*Chapter 25: Analog-to-Digital Converter (ADC)* Renamed *Section : Analog watchdog functionality* (was "Analog watchdog pulse width modulation bus") and reworded the section *Table 473: CDR field descriptions*, removed improper cross-references to conditional content for CDATA field *Section 25.3.9: Power-down mode*, replaced sentence: "If the CTU is enabled and the CSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set." with "If the CTU is enabled and the MSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set. *Chapter 26: Cross Triggering Unit (CTU)* *Section 26.4.3: ADC results*, replaced "FIFO1 and FIFO2—4 entries" with "FIFO1 and FIFO2—16 entries" *Section 26.8.21: Cross triggering unit control register (CTUCR)*, changed CRU_ADC_R to CTU_ADC_R *Table 502: CTUCR field descriptions*, added notes for DFE, CGRE, GRE, and TGSISR_RE bit fileds. *Chapter 28: eTimer* *Figure 550: DMA Request 0 Select register (DREQ0)*, changed bit 0 to DREW0_EN *Figure 551: DMA Request 1 Select register (DREQ1)*, changed bit 0 to DREW1_EN *Table 531: DREQn field descriptions*, added DREW1_EN description *Section : Comparator Load register 1 (CMPLD1)*, modified the description of CMPLD field to read, "Specifies the preload value for the COMP1 register" instead of "Specifies the preload value for the COMP2 register" *Chapter 29: Functional Safety* *Section : SWT Control Register (SWT_CR)*, specified reset value for SWT_1. *Chapter 30: Fault Collection and Control Unit (FCCU)* *Table 547: FCCU memory map*, in the access description of FCCU_CFG_TO register, changed "R always; W in CONFIG state only" to "R always; W always except CONFIG state" *Chapter 33: System Timer Module (STM)* *Table 587: STM memory map*, updated offset values of STM_CCR1, STM_CIR1 and STM_CMP1 *Chapter 35: Boot Assist Module (BAM)* *Table 600: Hardware configuration to select boot mode*: rewrote the column title from "ABS[2,0]" to ABS[1:0] removed footnote stating ABS[1] is "don't care" added footnote, "During reset the boot configuration pins are weak pull down. *Table 601: SPC56xP60/54 boot pins*, changed the function of A[3], from ABS[2] to ABS[1]. *Section 35.5.1: Entering boot modes*: Added PAD A[2] note Added the Boot Configuration Pins Added *Section 35.5.2: SPC56xP60/54 boot pins* Updated *Section : Configuration* with new *Figure 648: BAM Autoscan code flow* *Section : Boot from FlexCAN with autobaud enabled*, updated "UART boot mode" with "FlexCAN boot mode". Added *Section 35.7: Censorship*

Table 635. Revision history (continued)

Date	Revision	Substantive Changes
06-Jun-2012	3 cont'd	*Chapter 36: Voltage Regulators and Power Supplies* Updated *Section 36.1.1: High Power or Main Regulator (HPREG)* *Figure 655: Voltage Regulator Control register (VREG_CTL)*, changed address value to 0xC3FE_8080 *Figure 656: Voltage Regulator Status register (VREG_STATUS)*, changed address value to 0xC3FE_8084 *Chapter 37: IEEE 1149.1 Test Access Port Controller (JTAGC)* *Section 37.7.4: Boundary scan register*, replaced the sentence "The size of the boundary scan register is 464 bits." with "The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file."
18-Sep-2013	4	Updated Disclaimer.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com