RM0017
Reference manual

SPC560B40x, SPC560B50x, SPC560C40x, SPC560C50x 32-bit MCU family built on the embedded Power Architecture®

Introduction

The SPC560Bx and SPC560Cx is a new family of next generation microcontrollers built on the Power Architecture® embedded category. This document describes the features of the family and options available within the family members, and highlights important electrical and physical characteristics of the device.

The SPC560Bx and SPC560Cx family of 32-bit microcontrollers is the latest achievement in integrated automotive body application controllers. It belongs to an expanding family of automotive-focused products designed to address the next wave of body electronics applications within the vehicle. The advanced and cost-efficient host processor core of the SPC560Bx and SPC560Cx automotive controller family complies with the Power Architecture embedded category, which is 100 percent user-mode compatible with the original Power Architecture technology. It operates at speeds of up to 64 MHz and offers high performance processing optimized for low power consumption. It capitalizes on the available development infrastructure of current Power Architecture devices and is supported with software drivers, operating systems and configuration code to assist with users implementations.

Contents

1 Preface 37

1.1 Overview 37

1.2 Audience 37

1.3 Guide to this reference manual 37

1.4 Register description conventions 40

1.5 References 41

1.6 How to use the SPC560Bx and SPC560Cx documents 41

1.6.1 The SPC560Bx and SPC560Cx document set 41

1.6.2 Reference manual content 42

1.7 Using the SPC560Bx and SPC560Cx 43

1.7.1 Hardware design 43

1.7.2 Input/output pins 44

1.7.3 Software design 44

1.7.4 Other features 45

2 Introduction 46

2.1 The SPC560Bx and SPC560Cx microcontroller family 46

2.2 Features 46

2.2.1 SPC560Bx and SPC560Cx family comparison 46

2.2.2 Block diagram 49

2.2.3 Chip-level features 50

2.3 Packages 51

2.4 Developer support 51

3 Memory Map 52

4 Signal description 55

4.1 Introduction 55

4.2 Package pinouts 55

4.3 Pad configuration during reset phases 58

4.4 Voltage supply pins 59

4.5 Pad types 60

4.6 System pins 60

4.7 Functional ports 60
4.8 Nexus 2+ pins 78

5 Microcontroller Boot 79
5.1 Boot mechanism 79
5.1.1 Flash memory boot 80
5.1.2 Serial boot mode 82
5.1.3 Censorship 83
5.2 Boot Assist Module (BAM) 87
5.2.1 BAM software flow 87
5.2.2 LINFlex (RS232) boot 95
5.2.3 FlexCAN boot 96
5.3 System Status and Configuration Module (SSCM) 98
5.3.1 Introduction 98
5.3.2 Features 98
5.3.3 Modes of operation 99
5.3.4 Memory map and register description 99

6 Clock Description. 106
6.1 Clock architecture 106
6.2 Clock gating 107
6.3 Fast external crystal oscillator (FXOSC) digital interface 108
6.3.1 Main features 108
6.3.2 Functional description 108
6.3.3 Register description 109
6.4 Slow external crystal oscillator (SXOSC) digital interface 110
6.4.1 Introduction 110
6.4.2 Main features 110
6.4.3 Functional description 110
6.4.4 Register description 111
6.5 Slow internal RC oscillator (SIRC) digital interface 112
6.5.1 Introduction 112
6.5.2 Functional description 112
6.5.3 Register description 113
6.6 Fast internal RC oscillator (FIRC) digital interface 114
6.6.1 Introduction 114

6.6.2 Functional description 114
6.6.3 Register description 115
6.7 Frequency-modulated phase-locked loop (FMPLL) 115
6.7.1 Introduction 115
6.7.2 Overview 115
6.7.3 Features 116
6.7.4 Memory map 116
6.7.5 Register description 117
6.7.6 Functional description 120
6.7.7 Recommendations 123
6.8 Clock monitor unit (CMU) 123
6.8.1 Introduction 123
6.8.2 Main features 123
6.8.3 Block diagram 124
6.8.4 Functional description 125
6.8.5 Memory map and register description 126

7 Clock Generation Module (MC_CGM). 131
7.1 Overview 131
7.2 Features 133
7.3 Modes of Operation 133
7.3.1 Normal and Reset Modes of Operation 133
7.4 External Signal Description 133
7.5 Memory Map and Register Definition 133
7.5.1 Register Descriptions 137
7.6 Functional Description 141
7.6.1 System Clock Generation 141
7.6.2 Output Clock Multiplexing 141
7.6.3 Output Clock Division Selection 143

8 Mode Entry Module (MC_ME) 144
8.1 Introduction 144
8.1.1 Overview 144
8.1.2 Features 146
8.1.3 Modes of Operation 146
8.2 External Signal Description 147

8.3 Memory Map and Register Definition 147
8.3.1 Register Description 154
8.4 Functional Description 176
8.4.1 Mode Transition Request 176
8.4.2 Modes Details 178
8.4.3 Mode Transition Process 182
8.4.4 Protection of Mode Configuration Registers 191
8.4.5 Mode Transition Interrupts 191
8.4.6 Peripheral Clock Gating 193
8.4.7 Application Example 193

9 Reset Generation Module (MC_RGM). 195
9.1 Introduction 195
9.1.1 Overview 195
9.1.2 Features 197
9.1.3 Modes of operation 197
9.2 External signal description 198
9.3 Memory map and register definition 198
9.3.1 Register descriptions 200
9.4 Functional Description 212
9.4.1 Reset State Machine 212
9.4.2 Destructive Resets 216
9.4.3 External Reset 216
9.4.4 Functional Resets 217
9.4.5 STANDBY Entry Sequence 217
9.4.6 Alternate Event Generation 217
9.4.7 Boot Mode Capturing 218

10 Power Control Unit (MC_PCU) 219
10.1 Introduction 219
10.1.1 Overview 219
10.1.2 Features 220
10.1.3 Modes of Operation 220
10.2 External Signal Description 221
10.3 Memory Map and Register Definition 221
10.3.1 Register Descriptions 222

10.4 Functional Description 225
10.4.1 General 225
10.4.2 Reset / Power-On Reset 225
10.4.3 MC_PCU Configuration 225
10.4.4 Mode Transitions 226
10.5 Initialization Information 228
10.6 Application Information 228
10.6.1 STANDBY Mode Considerations 228

11 Voltage Regulators and Power Supplies 229
11.1 Voltage regulators 229
11.1.1 High power regulator (HPREG) 229
11.1.2 Low power regulator (LPREG) 229
11.1.3 Ultra low power regulator (ULPREG) 229
11.1.4 LVDs and POR 230
11.1.5 VREG digital interface 230
11.1.6 Register description 230
11.2 Power supply strategy 231
11.3 Power domain organization 232

12 Wakeup Unit (WKPU) 234
12.1 Overview 234
12.2 Features 236
12.3 External signal description 236
12.4 Memory map and register description 236
12.4.1 Memory map 236
12.4.2 NMI Status Flag Register (NSR) 237
12.4.3 NMI Configuration Register (NCR) 238
12.4.4 Wakeup/Interrupt Status Flag Register (WISR) 239
12.4.5 Interrupt Request Enable Register (IRER) 240
12.4.6 Wakeup Request Enable Register (WRER) 240
12.4.7 Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER) . . . 241
12.4.8 Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER) . . . 241
12.4.9 Wakeup/Interrupt Filter Enable Register (WIFER) 241
12.4.10 Wakeup/Interrupt Pullup Enable Register (WIPUER) 242
12.5 Functional description 242

12.5.1 General 242
12.5.2 Non-maskable interrupts 243
12.5.3 External wakeups/interrupts 244
12.5.4 On-chip wakeups 245

13 Real Time Clock / Autonomous Periodic Interrupt (RTC/API) 247
13.1 Overview 247
13.2 Features 247
13.3 Device-specific information 249
13.4 Modes of operation 249
13.4.1 Functional mode 249
13.4.2 Debug mode 250
13.5 Register descriptions 250
13.5.1 RTC Supervisor Control Register (RTCSUPV) 250
13.5.2 RTC Control Register (RTCC) 251
13.5.3 RTC Status Register (RTCS) 253
13.5.4 RTC Counter Register (RTCCNT) 254
13.6 RTC functional description 254
13.7 API functional description 255

14 CAN Sampler 256
14.1 Introduction 256
14.2 Main features 256
14.3 Register description 257
14.3.1 Control Register (CR) 257
14.3.2 Sample register n (n = 0..11) 258
14.4 Functional description 259
14.4.1 Enabling/Disabling the CAN sampler 259
14.4.2 Baud rate generation 260

15 e200z0h Core 261
15.1 Overview 261
15.2 Microarchitecture summary 261
15.3 Block diagram 263
15.4 Features 264
15.4.1 Instruction unit features 264

15.4.2 Integer unit features . . . 265
15.4.3 Load/Store unit features . . . 265
15.4.4 e200z0h system bus features . . . 265
15.4.5 Nexus 2+ features . . . 265
15.5 Core registers and programmer's model . . . 266

16 Interrupt Controller (INTC) . . . 268
16.1 Introduction . . . 268
16.2 Features . . . 268
16.3 Block diagram . . . 269
16.4 Modes of operation . . . 270
16.4.1 Normal mode . . . 270
16.5 Memory map and register description . . . 271
16.5.1 Module memory map . . . 271
16.5.2 Register description . . . 272
16.6 Functional description . . . 279
16.6.1 Interrupt request sources . . . 287
16.6.2 Priority management . . . 288
16.6.3 Handshaking with processor . . . 289
16.7 Initialization/application information . . . 291
16.7.1 Initialization flow . . . 291
16.7.2 Interrupt exception handler . . . 292
16.7.3 ISR, RTOS, and task hierarchy . . . 293
16.7.4 Order of execution . . . 294
16.7.5 Priority ceiling protocol . . . 295
16.7.6 Selecting priorities according to request rates and deadlines . . . 296
16.7.7 Software configurable interrupt requests . . . 296
16.7.8 Lowering priority within an ISR . . . 297
16.7.9 Negating an interrupt request outside of its ISR . . . 297
16.7.10 Examining LIFO contents . . . 298

17 Crossbar Switch (XBAR) . . . 299
17.1 Introduction . . . 299
17.2 Block diagram . . . 299
17.3 Overview . . . 300
17.4 Features . . . 300

17.5 Modes of operation 300
17.5.1 Normal mode 300
17.5.2 Debug mode 300
17.6 Functional description 300
17.6.1 Overview 300
17.6.2 General operation 301
17.6.3 Master ports 301
17.6.4 Slave ports 302
17.6.5 Priority assignment 302
17.6.6 Arbitration 302

18 Memory Protection Unit (MPU) 304
18.1 Introduction 304
18.2 Features 305
18.3 Modes of operation 306
18.4 External signal description 306
18.5 Memory map and register description 306
18.5.1 Memory map 307
18.5.2 Register description 308
18.6 Functional description 319
18.6.1 Access evaluation macro 319
18.6.2 Putting it all together and AHB error terminations 321
18.7 Initialization information 321
18.8 Application information 322

19 System Integration Unit Lite (SIUL) 323
19.1 Introduction 323
19.2 Overview 323
19.3 Features 325
19.4 External signal description 325
19.4.1 Detailed signal descriptions 326
19.5 Memory map and register description 327
19.5.1 SIUL memory map 327
19.5.2 Register protection 328
19.5.3 Register descriptions 328

19.6 Functional description . . . 345
19.6.1 Pad control . . . 345
19.6.2 General purpose input and output pads (GPIO) . . . 346
19.6.3 External interrupts . . . 347
19.7 Pin muxing . . . 348

20 Inter-Integrated Circuit Bus Controller Module (I^2C) . . . 349
20.1 Introduction . . . 349
20.1.1 Overview . . . 349
20.1.2 Features . . . 349
20.1.3 Block diagram . . . 349
20.2 External signal description . . . 350
20.2.1 SCL . . . 350
20.2.2 SDA . . . 350
20.3 Memory map and register description . . . 350
20.3.1 Module memory map . . . 350
20.3.2 I^2C Bus Address Register (IBAD) . . . 351
20.3.3 I^2C Bus Frequency Divider Register (IBFD) . . . 352
20.3.4 I^2C Bus Control Register (IBCR) . . . 358
20.3.5 I^2C Bus Status Register (IBSR) . . . 359
20.3.6 I^2C Bus Data I/O Register (IBDR) . . . 360
20.3.7 I^2C Bus Interrupt Config Register (IBIC) . . . 361
20.4 Functional description . . . 361
20.4.1 I-Bus protocol . . . 361
20.4.2 Interrupts . . . 365
20.5 Initialization/application information . . . 365
20.5.1 I^2C programming examples . . . 365

21 LIN Controller (LINFlex) . . . 371
21.1 Introduction . . . 371
21.2 Main features . . . 371
21.2.1 LIN mode features . . . 371
21.2.2 UART mode features . . . 371
21.2.3 Features common to LIN and UART . . . 372
21.3 General description . . . 372
21.4 Fractional baud rate generation . . . 373

21.5 Operating modes . . . 375
21.5.1 Initialization mode . . . 375
21.5.2 Normal mode . . . 375
21.5.3 Low power mode (Sleep) . . . 375
21.6 Test modes . . . 376
21.6.1 Loop Back mode . . . 376
21.6.2 Self Test mode . . . 376
21.7 Memory map and registers description . . . 377
21.7.1 Memory map . . . 377
21.8 Functional description . . . 403
21.8.1 UART mode . . . 403
21.8.2 LIN mode . . . 405
21.8.3 8-bit timeout counter . . . 413
21.8.4 Interrupts . . . 415

22 FlexCAN . . . 416
22.1 Introduction . . . 416
22.1.1 Overview . . . 416
22.1.2 FlexCAN module features . . . 417
22.1.3 Modes of operation . . . 418
22.2 External signal description . . . 418
22.2.1 Overview . . . 418
22.2.2 Signal descriptions . . . 419
22.3 Memory map and register description . . . 419
22.3.1 FlexCAN memory mapping . . . 419
22.3.2 Message buffer structure . . . 421
22.3.3 Rx FIFO structure . . . 424
22.3.4 Register description . . . 426
22.4 Functional description . . . 445
22.4.1 Overview . . . 445
22.4.2 Local priority transmission . . . 446
22.4.3 Transmit process . . . 446
22.4.4 Arbitration process . . . 447
22.4.5 Receive process . . . 447
22.4.6 Matching process . . . 449
22.4.7 Data coherence . . . 450

22.4.8 Rx FIFO . . . 452
22.4.9 CAN protocol related features . . . 453
22.4.10 Modes of operation details . . . 457
22.4.11 Interrupts . . . 458
22.4.12 Bus interface . . . 459
22.5 Initialization/Application information . . . 459
22.5.1 FlexCAN initialization sequence . . . 459
22.5.2 FlexCAN addressing and SRAM size configurations . . . 460

23 Deserial Serial Peripheral Interface (DSPI) . . . 461
23.1 Introduction . . . 461
23.2 Features . . . 462
23.3 Modes of operation . . . 463
23.3.1 Master mode . . . 463
23.3.2 Slave mode . . . 463
23.3.3 Module Disable mode . . . 463
23.3.4 Debug mode . . . 464
23.4 External signal description . . . 464
23.4.1 Signal overview . . . 464
23.4.2 Signal names and descriptions . . . 464
23.5 Memory map and register description . . . 465
23.5.1 Memory map . . . 465
23.5.2 DSPI Module Configuration Register (DSPIx_MCR) . . . 466
23.5.3 DSPI Transfer Count Register (DSPIx_TCR) . . . 469
23.5.4 DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn) . . . 470
23.5.5 DSPI Status Register (DSPIx_SR) . . . 478
23.5.6 DSPI Interrupt Request Enable Register (DSPIx_RSER) . . . 480
23.5.7 DSPI PUSH TX FIFO Register (DSPIx_PUSHR) . . . 482
23.5.8 DSPI POP RX FIFO Register (DSPIx_POPR) . . . 484
23.5.9 DSPI Transmit FIFO Registers 0–3 (DSPIx_TXFRn) . . . 485
23.6 Functional description . . . 486
23.6.1 Modes of operation . . . 487
23.6.2 Start and stop of DSPI transfers . . . 488
23.6.3 Serial peripheral interface (SPI) configuration . . . 489
23.6.4 DSPI baud rate and clock delay generation . . . 492
23.6.5 Transfer formats . . . 495

23.6.6 Continuous serial communications clock . . . 503
23.6.7 Interrupt requests . . . 506
23.6.8 Power saving features . . . 507
23.7 Initialization and application information . . . 508
23.7.1 How to change queues . . . 508
23.7.2 Baud rate settings . . . 508
23.7.3 Delay settings . . . 510
23.7.4 Calculation of FIFO pointer addresses . . . 510

24 Timers . . . 513
24.1 Introduction . . . 513
24.2 Technical overview . . . 513
24.2.1 Overview of the STM . . . 515
24.2.2 Overview of the eMIOS . . . 515
24.2.3 Overview of the PIT . . . 517
24.3 System Timer Module (STM) . . . 517
24.3.1 Introduction . . . 517
24.3.2 External signal description . . . 518
24.3.3 Memory map and register definition . . . 518
24.3.4 Functional description . . . 521
24.4 Enhanced Modular IO Subsystem (eMIOS) . . . 522
24.4.1 Introduction . . . 522
24.4.2 External signal description . . . 525
24.4.3 Memory map and register description . . . 525
24.4.4 Functional description . . . 537
24.4.5 Initialization/Application information . . . 567
24.5 Periodic Interrupt Timer (PIT) . . . 571
24.5.1 Introduction . . . 571
24.5.2 Features . . . 571
24.5.3 Signal description . . . 572
24.5.4 Memory map and register description . . . 572
24.5.5 Functional description . . . 576
24.5.6 Initialization and application information . . . 577

25 Analog-to-Digital Converter (ADC) . . . 579
25.1 Overview . . . 579

25.1.1 Device-specific features . . . 579
25.1.2 Device-specific implementation . . . 580
25.2 Introduction . . . 580
25.3 Functional description . . . 581
25.3.1 Analog channel conversion . . . 581
25.3.2 Analog clock generator and conversion timings . . . 584
25.3.3 ADC sampling and conversion timing . . . 585
25.3.4 ADC CTU (Cross Triggering Unit) . . . 586
25.3.5 Presampling . . . 587
25.3.6 Programmable analog watchdog . . . 588
25.3.7 Interrupts . . . 590
25.3.8 External decode signals delay . . . 590
25.3.9 Power-down mode . . . 590
25.3.10 Auto-clock-off mode . . . 591
25.4 Register descriptions . . . 591
25.4.1 Introduction . . . 591
25.4.2 Control logic registers . . . 596
25.4.3 Interrupt registers . . . 600
25.4.4 Threshold registers . . . 607
25.4.5 Presampling registers . . . 608
25.4.6 Conversion timing registers CTR[0..2] . . . 611
25.4.7 Mask registers . . . 611
25.4.8 Delay registers . . . 616
25.4.9 Data registers . . . 617

26 Cross Triggering Unit (CTU) . . . 619
26.1 Introduction . . . 619
26.2 Main features . . . 619
26.3 Block diagram . . . 619
26.4 Memory map and register descriptions . . . 620
26.4.1 Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63) . . . 620
26.5 Functional description . . . 621
26.5.1 Channel value . . . 623

27 Static RAM (SRAM) . . . 625
27.1 Introduction . . . 625

27.2 Low power configuration 625
27.3 Register memory map 625
27.4 SRAM ECC mechanism 626
27.4.1 Access timing 626
27.4.2 Reset effects on SRAM accesses 627
27.5 Functional description 627
27.6 Initialization and application information 627

28 Flash Memory 629
28.1 Introduction 629
28.2 Main features 630
28.3 Block diagram 630
28.4 Functional description 631
28.4.1 Module structure 631
28.4.2 Flash memory module sectorization 632
28.4.3 TestFlash block 633
28.4.4 Shadow sector 634
28.4.5 User mode operation 635
28.4.6 Reset 636
28.4.7 Power-down mode 636
28.4.8 Low power mode 637
28.5 Register description 637
28.5.1 CFlash register description 639
28.5.2 DFlash register description 668
28.6 Programming considerations 692
28.6.1 Modify operation 692
28.6.2 Double word program 693
28.6.3 Sector erase 695
28.7 Platform flash memory controller 703
28.7.1 Introduction 703
28.7.2 Memory map and register description 706
28.8 Functional description 715
28.8.1 Access protections 716
28.8.2 Read cycles – Buffer miss 716
28.8.3 Read cycles – Buffer hit 716
28.8.4 Write cycles 716

28.8.5 Error termination 716
28.8.6 Access pipelining 717
28.8.7 Flash error response operation 717
28.8.8 Bank0 page read buffers and prefetch operation 717
28.8.9 Bank1 Temporary Holding Register 719
28.8.10 Read-while-write functionality 720
28.8.11 Wait-state emulation 721

29 Register Protection 723
29.1 Introduction 723
29.2 Features 723
29.3 Modes of operation 724
29.4 External signal description 724
29.5 Memory map and register description 724
29.5.1 Memory map 725
29.5.2 Register description 726
29.6 Functional description 728
29.6.1 General 728
29.6.2 Change lock settings 728
29.6.3 Access errors 731
29.7 Reset 732
29.8 Protected registers 732

30 Software Watchdog Timer (SWT) 737
30.1 Overview 737
30.2 Features 737
30.3 Modes of operation 737
30.4 External signal description 737
30.5 Memory map and register description 738
30.5.1 Memory map 738
30.5.2 Register description 738
30.6 Functional description 743

31 Error Correction Status Module (ECSM) 745
31.1 Introduction 745

31.2 Overview . . . 745
31.3 Features . . . 745
31.4 Memory map and register description . . . 745
31.4.1 Memory map . . . 745
31.4.2 Register description . . . 746
31.4.3 Register protection . . . 767

32 IEEE 1149.1 Test Access Port Controller (JTAGC) . . . 768
32.1 Introduction . . . 768
32.2 Block diagram . . . 768
32.3 Overview . . . 768
32.4 Features . . . 769
32.5 Modes of operation . . . 769
32.5.1 Reset . . . 769
32.5.2 IEEE 1149.1-2001 defined test modes . . . 769
32.6 External signal description . . . 770
32.7 Memory map and register description . . . 770
32.7.1 Instruction Register . . . 770
32.7.2 Bypass Register . . . 771
32.7.3 Device Identification Register . . . 771
32.7.4 Boundary Scan Register . . . 772
32.8 Functional Description . . . 772
32.8.1 JTAGC Reset Configuration . . . 772
32.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port . . . 772
32.8.3 TAP controller state machine . . . 772
32.8.4 JTAGC instructions . . . 774
32.8.5 Boundary Scan . . . 776
32.9 e200z0 OnCE controller . . . 776
32.9.1 e200z0 OnCE Controller Block Diagram . . . 776
32.9.2 e200z0 OnCE Controller Functional Description . . . 777
32.9.3 e200z0 OnCE Controller Register Description . . . 777
32.10 Initialization/application information . . . 779

33 Nexus Development Interface (NDI) . . . 780
33.1 Introduction . . . 780
33.2 Block diagram . . . 780

33.3 Features 781
33.4 Modes of Operation 782
33.4.1 Nexus Reset 782
33.4.2 Operating Mode 783
33.5 External Signal Description 783
33.5.1 Nexus Signal Reset States 783
33.6 Memory Map and Register Description 783
33.6.1 Nexus Debug Interface Registers 784
33.6.2 Register Description 785
33.7 Functional description 795
33.7.1 NPC_HNDSHK module 795
33.7.2 Enabling Nexus Clients for TAP Access 796
33.7.3 Configuring the NDI for Nexus Messaging 796
33.7.4 Programmable MCKO Frequency 797
33.7.5 Nexus Messaging 797
33.7.6 EVTO Sharing 797
33.7.7 Debug Mode Control 798
33.7.8 Ownership Trace 798

Appendix A Register Map 801

Revision history 879

List of tables

Table 1. Guide to this reference manual . . . 37
Table 2. Reference manual integration and functional content . . . 42
Table 3. Code Flash memory scaling . . . 46
Table 4. SRAM memory scaling . . . 46
Table 5. SPC560Bx and SPC560Cx device comparison . . . 47
Table 6. SPC560Bx and SPC560Cx memory map . . . 52
Table 7. Voltage supply pin descriptions . . . 59
Table 8. System pin descriptions . . . 60
Table 9. Functional port pin descriptions . . . 61
Table 10. Nexus 2+ pin descriptions . . . 78
Table 11. Boot mode selection . . . 79
Table 12. RCHW field descriptions . . . 81
Table 13. Examples of legal and illegal passwords . . . 83
Table 14. Censorship configuration and truth table . . . 84
Table 15. SSCM_STATUS[BMODE] values as used by BAM . . . 89
Table 16. Serial boot mode – baud rates . . . 89
Table 17. BAM censorship mode detection . . . 90
Table 18. UART boot mode download protocol . . . 95
Table 19. FlexCAN boot mode download protocol . . . 97
Table 20. SSCM memory map . . . 99
Table 21. SSCM_STATUS allowed register accesses . . . 99
Table 22. SSCM_STATUS field descriptions . . . 100
Table 23. SSCM_MEMCONFIG field descriptions . . . 101
Table 24. SSCM_MEMCONFIG allowed register accesses . . . 101
Table 25. SSCM_ERROR field descriptions . . . 102
Table 26. SSCM_ERROR allowed register accesses . . . 102
Table 27. SSCM_DEBUGPORT field descriptions . . . 103
Table 28. Debug status port modes . . . 103
Table 29. SSCM_DEBUGPORT allowed register accesses . . . 103
Table 30. Password Comparison Register field descriptions . . . 104
Table 31. SSCM_PWCMPH/L allowed register accesses . . . 105
Table 32. SPC560Bx and SPC560Cx — Peripheral clock sources . . . 108
Table 33. Truth table of crystal oscillator . . . 109
Table 34. FXOSC_CTL field descriptions . . . 110
Table 35. SXOSC truth table . . . 111
Table 36. SXOSC_CTL field descriptions . . . 112
Table 37. SIRC_CTL field descriptions . . . 113
Table 38. FIRC_CTL field descriptions . . . 115
Table 39. FMPLL memory map . . . 117
Table 40. CR field descriptions . . . 117
Table 41. Input divide ratios . . . 118
Table 42. Output divide ratios . . . 119
Table 43. Loop divide ratios . . . 119
Table 44. MR field descriptions . . . 120
Table 45. FMPLL lookup table . . . 121
Table 46. Progressive clock switching on pll_select rising edge . . . 121
Table 47. CMU memory map . . . 126
Table 48. CMU_CSR field descriptions . . . 127

Table 49. CMU_FDR field descriptions . . . 128
Table 50. CMU_HFREFR field descriptions . . . 128
Table 51. CMU_LFREFR field descriptions . . . 129
Table 52. CMU_ISR field descriptions . . . 129
Table 53. CMU_MDR field descriptions . . . 130
Table 54. MC_CGM Register Description . . . 133
Table 55. MC_CGM Memory Map . . . 134
Table 56. Output Clock Enable Register (CGM_OC_EN) Field Descriptions . . . 138
Table 57. Output Clock Division Select Register (CGM_OCDS_SC) Field Descriptions . . . 139
Table 58. System Clock Select Status Register (CGM_SC_SS) Field Descriptions . . . 140
Table 59. System Clock Divider Configuration Registers (CGM_SC_DC0…2) Field Descriptions . 140
Table 60. MC_ME Mode Descriptions . . . 146
Table 61. MC_ME Register Description . . . 147
Table 62. MC_ME Memory Map . . . 150
Table 63. Global Status Register (ME_GS) Field Descriptions . . . 155
Table 64. Mode Control Register (ME_MCTL) Field Descriptions . . . 158
Table 65. Mode Enable Register (ME_ME) Field Descriptions . . . 159
Table 66. Interrupt Status Register (ME_IS) Field Descriptions . . . 160
Table 67. Interrupt Mask Register (ME_IM) Field Descriptions . . . 161
Table 68. Invalid Mode Transition Status Register (ME_IMTS) Field Descriptions . . . 162
Table 69. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions . . . 163
Table 70. Mode Configuration Registers (ME_<mode>_MC) Field Descriptions . . . 169
Table 71. Peripheral Status Registers 0…4 (ME_PS0…4) Field Descriptions . . . 173
Table 72. Run Peripheral Configuration Registers (ME_RUN_PC0…7) Field Descriptions . . . 173
Table 73. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) Field Descriptions . . . 174
Table 74. Peripheral Control Registers (ME_PCTL0…143) Field Descriptions . . . 175
Table 75. Peripheral control registers by peripheral . . . 175
Table 76. MC_ME Resource Control Overview . . . 183
Table 77. MC_ME System Clock Selection Overview . . . 187
Table 78. MC_RGM register description . . . 198
Table 79. MC_RGM Memory Map . . . 199
Table 80. Functional Event Status Register (RGM_FES) Field Descriptions . . . 201
Table 81. Destructive Event Status Register (RGM_DES) Field Descriptions . . . 203
Table 82. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions . . . 204
Table 83. Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions . . . 206
Table 84. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions . . . 207
Table 85. Destructive Event Alternate Request Register (RGM_DEAR) Field Descriptions . . . 208
Table 86. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions . . . 209
Table 87. STANDBY Reset Sequence Register (RGM_STDBY) Field Descriptions . . . 211
Table 88. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions211
Table 89. MC_RGM Reset Implications . . . 212
Table 90. MC_RGM Alternate Event Selection . . . 218
Table 91. MC_PCU Register Description . . . 221
Table 92. MC_PCU Memory Map . . . 221
Table 93. Power Domain Configuration Register Field Descriptions . . . 223
Table 94. Power Domain Status Register (PCU_PSTAT) Field Descriptions . . . 225
Table 95. VREG_CTL field descriptions . . . 231
Table 96. Wakeup vector mapping . . . 234
Table 97. WKPU memory map . . . 237
Table 98. NSR field descriptions . . . 238
Table 99. NCR field descriptions . . . 238

Table 100. WISR field descriptions . . . 239
Table 101. IRER field descriptions . . . 240
Table 102. WRER field descriptions . . . 240
Table 103. WIREER field descriptions . . . 241
Table 104. WIFEER field descriptions . . . 241
Table 105. WIFER field descriptions . . . 242
Table 106. WIPUER field descriptions . . . 242
Table 107. RTC/API register map . . . 250
Table 108. RTCSUPV field descriptions . . . 250
Table 109. RTCC field descriptions . . . 251
Table 110. RTCS field descriptions . . . 253
Table 111. RTCCNTfield descriptions . . . 254
Table 112. CAN sampler memory map . . . 257
Table 113. CR field descriptions . . . 257
Table 114. Interrupt sources available . . . 269
Table 115. INTC memory map . . . 272
Table 116. INTC_MCR field descriptions . . . 273
Table 117. INTC_CPR field descriptions . . . 273
Table 118. PRI values . . . 274
Table 119. INTC_IACKR field descriptions . . . 275
Table 120. INTC_SSCIR[0:7] field descriptions . . . 277
Table 121. INTC_PSR0_3–INTC_PSR208_210 field descriptions . . . 278
Table 122. INTC Priority Select Register address offsets . . . 278
Table 123. Interrupt vector table . . . 280
Table 124. Order of ISR execution example . . . 294
Table 125. XBAR switch ports for SPC560Bx and SPC560Cx . . . 299
Table 126. Hardwired bus master priorities . . . 302
Table 127. MPU memory map . . . 307
Table 128. MPU_CESR field descriptions . . . 308
Table 129. MPU_EARn field descriptions . . . 309
Table 130. MPU_EDRn field descriptions . . . 310
Table 131. MPU_RGDn.Word0 field descriptions . . . 311
Table 132. MPU_RGDn.Word1 field descriptions . . . 312
Table 133. MPU_RGDn.Word2 field descriptions . . . 313
Table 134. MPU_RGDn.Word3 field descriptions . . . 316
Table 135. MPU_RGDAACn field descriptions . . . 317
Table 136. Protection violation definition . . . 320
Table 137. SIUL signal properties . . . 325
Table 138. SIUL memory map . . . 327
Table 139. MIDR1 field descriptions . . . 329
Table 140. MIDR2 field descriptions . . . 330
Table 141. ISR field descriptions . . . 331
Table 142. IRER field descriptions . . . 332
Table 143. IREER field descriptions . . . 333
Table 144. IFEER field descriptions . . . 333
Table 145. IFER field descriptions . . . 334
Table 146. PCR bit implementation by pad type . . . 335
Table 147. PCRx field descriptions . . . 335
Table 148. PSMI0_3 field descriptions . . . 337
Table 149. Peripheral input pin selection . . . 338
Table 150. GPDO0_3 field descriptions . . . 340
Table 151. GPDI0_3 field descriptions . . . 341

Table 152. PGPDO0 – PGPDO3 register map . . . 341
Table 153. PGPDI0 – PGPDI3 register map . . . 342
Table 154. MPGPDO0 – MPGPDO7 register map . . . 343
Table 155. MPGPDO0..MPGPDO7 field descriptions . . . 344
Table 156. IFMC field descriptions . . . 345
Table 157. IFCPR field descriptions . . . 345
Table 158. I2C memory map . . . 351
Table 159. IBAD field descriptions . . . 351
Table 160. IBFD field descriptions . . . 352
Table 161. I-Bus multiplier factor . . . 352
Table 162. I-Bus prescaler divider values . . . 352
Table 163. I-Bus tap and prescale values . . . 353
Table 164. I^2C divider and hold values . . . 355
Table 165. IBCR field descriptions . . . 358
Table 166. IBSR Field Descriptions . . . 359
Table 167. IBIC field descriptions . . . 361
Table 168. Interrupt summary . . . 365
Table 169. Error calculation for programmed baud rates . . . 374
Table 170. LINFlex memory map . . . 377
Table 171. LINCR1 field descriptions . . . 379
Table 172. Checksum bits configuration . . . 380
Table 173. LIN master break length selection . . . 380
Table 174. Operating mode selection . . . 381
Table 175. LINIER field descriptions . . . 381
Table 176. LINSR field descriptions . . . 384
Table 177. LINESR field descriptions . . . 386
Table 178. UARTCR field descriptions . . . 388
Table 179. UARTSR field descriptions . . . 389
Table 180. LINTCSR field descriptions . . . 391
Table 181. LINOCR field descriptions . . . 392
Table 182. LINTOCR field descriptions . . . 393
Table 183. LINFBRR field descriptions . . . 393
Table 184. LINIBRR field descriptions . . . 394
Table 185. Integer baud rate selection . . . 394
Table 186. LINCFR field descriptions . . . 395
Table 187. LINCR2 field descriptions . . . 396
Table 188. BIDR field descriptions . . . 397
Table 189. BDRL field descriptions . . . 398
Table 190. BDRM field descriptions . . . 399
Table 191. IFER field descriptions . . . 399
Table 192. IFER[FACT] configuration . . . 399
Table 193. IFMI field descriptions . . . 400
Table 194. IFMR field descriptions . . . 401
Table 195. IFMR[IFM] configuration . . . 401
Table 196. IFCR2*n* field descriptions . . . 402
Table 197. IFCR2*n* + 1 field descriptions . . . 403
Table 198. Message buffer . . . 404
Table 199. Filter to interrupt vector correlation . . . 411
Table 200. LINFlex interrupt control . . . 415
Table 201. FlexCAN signals . . . 419
Table 202. FlexCAN memory map . . . 420
Table 203. Message buffer MB0 memory mapping . . . 421

Table 204. Message Buffer Structure field description . . . 421
Table 205. Message buffer code for Rx buffers . . . 423
Table 206. Message buffer code for Tx buffers . . . 423
Table 207. Rx FIFO Structure field description . . . 426
Table 208. MCR field descriptions . . . 427
Table 209. IDAM coding . . . 430
Table 210. CTRL field descriptions . . . 431
Table 211. RXGMASK field description . . . 435
Table 212. ESR field descriptions . . . 438
Table 213. Fault confinement state . . . 440
Table 214. IMASK2 field descriptions . . . 441
Table 215. IMASK1 field descriptions . . . 442
Table 216. IFLAG2 field descriptions . . . 443
Table 217. IFLAG1 field descriptions . . . 444
Table 218. RXIMR0 – RXIMR63 field description . . . 445
Table 219. Time segment syntax . . . 456
Table 220. CAN standard compliant bit time segment settings . . . 456
Table 221. Minimum ratio between peripheral clock frequency and CAN bit rate . . . 457
Table 222. Signal properties . . . 464
Table 223. DSPI memory map . . . 465
Table 224. DSPIx_MCR field descriptions . . . 467
Table 225. DSPIx_TCR field descriptions . . . 470
Table 226. DSPI*x*_CTAR*n* field descriptions . . . 471
Table 227. DSPI SCK duty cycle . . . 474
Table 228. DSPI transfer frame size . . . 474
Table 229. DSPI PCS to SCK delay scaler . . . 474
Table 230. DSPI After SCK delay scaler . . . 475
Table 231. DSPI delay after transfer scaler . . . 475
Table 232. DSPI baud rate scaler . . . 475
Table 233. DSPI SCK duty cycle . . . 476
Table 234. DSPI transfer frame size . . . 476
Table 235. DSPI PCS to SCK delay scaler . . . 476
Table 236. DSPI After SCK delay scaler . . . 477
Table 237. DSPI delay after transfer scaler . . . 477
Table 238. DSPI baud rate scaler . . . 477
Table 239. DSPIx_SR field descriptions . . . 478
Table 240. DSPIx_RSER field descriptions . . . 481
Table 241. DSPIx_PUSHR field descriptions . . . 483
Table 242. DSPIx_POPR field descriptions . . . 484
Table 243. DSPIx_TXFRn field descriptions . . . 485
Table 244. DSPIx_RXFRn field description . . . 486
Table 245. State transitions for start and stop of DSPI transfers . . . 489
Table 246. Baud rate computation example . . . 493
Table 247. CS to SCK delay computation example . . . 493
Table 248. After SCK delay computation example . . . 494
Table 249. Delay after transfer computation example . . . 494
Table 250. Peripheral chip select strobe assert computation example . . . 495
Table 251. Peripheral chip select strobe negate computation example . . . 495
Table 252. Delayed master sample point . . . 499
Table 253. Interrupt request conditions . . . 506
Table 254. Baud rate values . . . 509
Table 255. Delay values . . . 510

Table 256. eMIOS_0 channel to pin mapping . . . 516
Table 257. eMIOS_1 channel to pin mapping . . . 516
Table 258. STM memory map . . . 518
Table 259. STM_CR field descriptions . . . 519
Table 260. STM_CNT field descriptions . . . 520
Table 261. STM_CCRn field descriptions . . . 520
Table 262. STM_CIRn field descriptions . . . 521
Table 263. STM_CMPn field descriptions . . . 521
Table 264. eMIOS memory map . . . 525
Table 265. Unified Channel memory map . . . 526
Table 266. EMIOSMCR field descriptions . . . 526
Table 267. Global prescaler clock divider . . . 527
Table 268. EMIOSGFLAG field descriptions . . . 528
Table 269. EMIOSOUDIS field descriptions . . . 528
Table 270. EMIOSUCDIS field descriptions . . . 529
Table 271. EMIOSA[n], EMIOSB[n] and EMIOSALTA[n] values assignment . . . 530
Table 272. EMIOSC[n] field descriptions . . . 532
Table 273. UC internalprescaler clock divider . . . 534
Table 274. UC input filter bits . . . 534
Table 275. UC BSL bits . . . 534
Table 276. Channel mode selection . . . 535
Table 277. EMIOSS[n] field descriptions . . . 536
Table 278. PIT memory map . . . 572
Table 279. Timer channel *n* . . . 572
Table 280. PITMCR field descriptions . . . 573
Table 281. LDVAL field descriptions . . . 574
Table 282. CVAL field descriptions . . . 574
Table 283. TCTRL field descriptions . . . 575
Table 284. TFLG field descriptions . . . 576
Table 285. Configurations for starting normal conversion . . . 581
Table 286. ADC sampling and conversion timing at 5 V / 3.3 V for ADC_0 . . . 586
Table 287. Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC_0 586
Table 288. Presampling voltage selection based on PREVALx fields . . . 588
Table 289. Values of WDGxH and WDGxL fields . . . 589
Table 290. ADC_0 digital registers . . . 591
Table 291. MCR field descriptions . . . 597
Table 292. MSR field descriptions . . . 599
Table 293. ISR field descriptions . . . 600
Table 294. CEOCFR field descriptions . . . 602
Table 295. Interrupt Mask Register (IMR) field descriptions . . . 603
Table 296. CIMR field descriptions . . . 604
Table 297. WTISR field descriptions . . . 605
Table 298. WTIMR field descriptions . . . 606
Table 299. TRCx field descriptions . . . 607
Table 300. THRHLRx field descriptions . . . 608
Table 301. PSCR field descriptions . . . 609
Table 302. PSR field descriptions . . . 610
Table 303. CTR field descriptions . . . 611
Table 304. NCMR field descriptions . . . 613
Table 305. JCMR field descriptions . . . 615
Table 306. DSDR field descriptions . . . 616
Table 307. PDEDR field descriptions . . . 617

Table 308. CDR field descriptions . . . 617
Table 309. CTU memory map . . . 620
Table 310. CTU_EVTCFGRx field descriptions . . . 620
Table 311. Trigger source . . . 621
Table 312. CTU-to-ADC channel assignment . . . 623
Table 313. Low power configuration . . . 625
Table 314. SRAM memory map . . . 625
Table 315. Number of wait states required for SRAM operations . . . 627
Table 316. Flash memory features . . . 630
Table 317. CFlash module sectorization . . . 632
Table 318. DFlash module sectorization . . . 633
Table 319. CFlash TestFlash structure . . . 633
Table 320. DFlash TestFlash structure . . . 634
Table 321. Shadow sector structure . . . 635
Table 322. CFlash registers . . . 637
Table 323. DFlash registers . . . 638
Table 324. CFLASH_MCR field descriptions . . . 639
Table 326. Low address space configuration . . . 643
Table 327. Mid address space configuration . . . 643
Table 325. Array space size . . . 643
Table 328. CFLASH_MCR bits set/clear priority levels . . . 644
Table 329. CFLASH_LML field descriptions . . . 645
Table 330. CFLASH_NVLML field descriptions . . . 647
Table 331. CFLASH_SLL field descriptions . . . 649
Table 332. CFLASH_NVSLL field descriptions . . . 652
Table 333. CFLASH_LMS field descriptions . . . 655
Table 334. CFLASH_ADR field descriptions . . . 656
Table 335. CFLASH_ADR content: priority list . . . 656
Table 336. CFLASH_UT0 field descriptions . . . 657
Table 337. CFLASH_UT1 field descriptions . . . 659
Table 338. CFLASH_UT2 field descriptions . . . 660
Table 339. CFLASH_UMISR0 field descriptions . . . 660
Table 340. CFLASH_UMISR1 field descriptions . . . 661
Table 341. CFLASH_UMISR2 field descriptions . . . 662
Table 342. CFLASH_UMISR3 field descriptions . . . 663
Table 343. CFLASH_UMISR4 field descriptions . . . 664
Table 344. NVPWD0 field descriptions . . . 665
Table 345. NVPWD1 field descriptions . . . 666
Table 346. NVSCC0 field descriptions . . . 666
Table 347. NVSCC1 field descriptions . . . 667
Table 348. NVUSRO field descriptions . . . 668
Table 349. DFLASH_MCR field descriptions . . . 669
Table 350. Array space size . . . 672
Table 351. Low address space configuration . . . 673
Table 352. Mid address space configuration . . . 673
Table 353. DFLASH_MCR bits set/clear priority levels . . . 673
Table 354. DFLASH_LML field descriptions . . . 675
Table 355. DFLASH_NVLML field descriptions . . . 677
Table 356. DFLASH_SLL field descriptions . . . 679
Table 357. DFLASH_NVSLL field descriptions . . . 681
Table 358. DFLASH_LMS field descriptions . . . 682
Table 359. DFLASH_ADR field descriptions . . . 683

Table 360. DFLASH_ADR content: priority list . . . 683
Table 361. DFLASH_UT0 field descriptions . . . 684
Table 362. DFLASH_UT1 field descriptions . . . 686
Table 363. DFLASH_UT2 field descriptions . . . 687
Table 364. DFLASH_UMISR0 field descriptions . . . 688
Table 365. DFLASH_UMISR1 field descriptions . . . 689
Table 366. DFLASH_UMISR2 field descriptions . . . 690
Table 367. DFLASH_UMISR3 field descriptions . . . 691
Table 368. DFLASH_UMISR4 field descriptions . . . 692
Table 369. Flash memory modify operations . . . 693
Table 370. Bit manipulation: Double words with the same ECC value . . . 701
Table 371. Flash memory-related regions in the system memory map . . . 707
Table 372. Platform flash memory controller 32-bit memory map . . . 707
Table 373. PFCR0 field descriptions . . . 709
Table 374. PFCR1 field descriptions . . . 712
Table 375. PFAPR field descriptions . . . 714
Table 376. NVPFAPR field descriptions . . . 715
Table 377. Platform flash memory controller stall-while-write interrupts . . . 721
Table 378. Additional wait-state encoding . . . 721
Table 379. Extended additional wait-state encoding . . . 722
Table 380. Register protection memory map . . . 725
Table 381. SLBR*n* field descriptions . . . 726
Table 382. Soft lock bits vs. protected address . . . 727
Table 383. GCR field descriptions . . . 728
Table 384. Protected registers . . . 732
Table 385. SWT memory map . . . 738
Table 386. SWT_CR field descriptions . . . 739
Table 387. SWT_IR field descriptions . . . 740
Table 388. SWT_TO Register field descriptions . . . 741
Table 389. SWT_WN Register field descriptions . . . 741
Table 390. SWT_SR field descriptions . . . 742
Table 391. SWT_CO field descriptions . . . 742
Table 392. ECSM memory map . . . 745
Table 393. PCT field descriptions . . . 747
Table 394. REV field descriptions . . . 747
Table 395. IOPMC field descriptions . . . 748
Table 396. MWCR field descriptions . . . 749
Table 397. MIR field descriptions . . . 750
Table 398. MUDCR field descriptions . . . 751
Table 399. ECR field descriptions . . . 753
Table 400. ESR field descriptions . . . 755
Table 401. EEGR field descriptions . . . 757
Table 402. PFEAR field descriptions . . . 759
Table 403. PFEMR field descriptions . . . 760
Table 404. PFEAT field descriptions . . . 761
Table 405. PFEDR field descriptions . . . 762
Table 406. PREAR field descriptions . . . 762
Table 407. PRESR field descriptions . . . 763
Table 408. RAM syndrome mapping for single-bit correctable errors . . . 763
Table 409. PREMR field descriptions . . . 765
Table 410. PREAT field descriptions . . . 766
Table 411. PREDR field descriptions . . . 766

Table 412. JTAG signal properties . . . 770
Table 413. Device Identification Register Field Descriptions . . . 771
Table 414. JTAG Instructions . . . 774
Table 415. e200z0 OnCE Register Addressing . . . 778
Table 416. NDI Signal Reset State . . . 783
Table 417. Nexus Debug Interface Registers . . . 784
Table 418. DID field descriptions . . . 785
Table 419. PCR field descriptions . . . 786
Table 420. DC1 field descriptions . . . 788
Table 421. DC2 field descriptions . . . 790
Table 422. DS field descriptions . . . 790
Table 423. RWCS field descriptions . . . 792
Table 424. Read/Write Access Status Bit Encoding . . . 793
Table 425. WT field descriptions . . . 795
Table 426. JTAGC Instruction Opcodes to Enable Nexus Clients . . . 796
Table 427. Nexus Client JTAG Instructions . . . 796
Table 428. NDI configuration options . . . 797
Table 429. SRC Packet Encodings . . . 797
Table 430. Error Code Encoding (TCODE = 8) . . . 799
Table 431. Module base addresses . . . 801
Table 432. Detailed register map . . . 802
Table 433. Document revision history . . . 879

List of figures

Figure 1. Register figure conventions . . . 41
Figure 2. SPC560Bx and SPC560Cx block diagram . . . 49
Figure 3. LQFP 64-pin configuration . . . 55
Figure 4. LQFP 100-pin configuration (top view) . . . 56
Figure 5. LQFP 144-pin configuration (top view) . . . 57
Figure 6. LBGA208 configuration . . . 58
Figure 7. Boot mode selection . . . 80
Figure 8. Boot sector structure . . . 81
Figure 9. Flash memory boot mode sequence . . . 82
Figure 10. Censorship control in flash memory boot mode . . . 86
Figure 11. Censorship control in serial boot mode . . . 87
Figure 12. BAM logic flow . . . 88
Figure 13. BAM censorship mode detection . . . 91
Figure 14. BAM serial boot mode flow for censorship enabled and private password . . . 93
Figure 15. Start address, VLE bit and download size in bytes . . . 94
Figure 16. LINFlex bit timing in UART mode . . . 95
Figure 17. FlexCAN bit timing . . . 97
Figure 18. SSCM block diagram . . . 98
Figure 19. System Status Register (SSCM_STATUS) . . . 99
Figure 20. System Memory Configuration Register (SSCM_MEMCONFIG) . . . 100
Figure 21. Error Configuration (SSCM_ERROR) . . . 101
Figure 22. Debug Status Port Register (SSCM_DEBUGPORT) . . . 102
Figure 23. Password Comparison Register High Word (SSCM_PWCMPH) . . . 104
Figure 24. Password Comparison Register Low Word (SSCM_PWCMPL) . . . 104
Figure 25. SPC560Bx and SPC560Cx system clock generation . . . 107
Figure 26. Fast External Crystal Oscillator Control Register (FXOSC_CTL) . . . 109
Figure 27. Slow External Crystal Oscillator Control Register (SXOSC_CTL) . . . 111
Figure 28. Low Power RC Control Register (SIRC_CTL) . . . 113
Figure 29. FIRC Oscillator Control Register (FIRC_CTL) . . . 115
Figure 30. FMPLL block diagram . . . 116
Figure 31. Control Register (CR) . . . 117
Figure 32. Modulation Register (MR) . . . 119
Figure 33. FMPLL output clock division flow during progressive switching . . . 121
Figure 34. Frequency modulation . . . 122
Figure 35. Clock Monitor Unit diagram . . . 124
Figure 36. Control Status Register (CMU_CSR) . . . 127
Figure 37. Frequency Display Register (CMU_FDR) . . . 128
Figure 38. High Frequency Reference Register FMPLL (CMU_HFREFR) . . . 128
Figure 39. Low Frequency Reference Register FMPLL (CMU_LFREFR) . . . 129
Figure 40. Interrupt status register (CMU_ISR) . . . 129
Figure 41. Measurement Duration Register (CMU_MDR) . . . 130
Figure 42. MC_CGM Block Diagram . . . 132
Figure 43. Output Clock Enable Register (CGM_OC_EN) . . . 138
Figure 44. Output Clock Division Select Register (CGM_OCDS_SC) . . . 138
Figure 45. System Clock Select Status Register (CGM_SC_SS) . . . 139
Figure 46. System Clock Divider Configuration Registers (CGM_SC_DC0...2) . . . 140
Figure 47. MC_CGM System Clock Generation Overview . . . 142
Figure 48. MC_CGM Output Clock Multiplexer and PA[0] Generation . . . 143

Figure 49. MC_MEBlock Diagram . . . 145
Figure 50. Global Status Register (ME_GS) . . . 155
Figure 51. Mode Control Register (ME_MCTL) . . . 157
Figure 52. Mode Enable Register (ME_ME) . . . 158
Figure 53. Interrupt Status Register (ME_IS) . . . 160
Figure 54. Interrupt Mask Register (ME_IM) . . . 161
Figure 55. Invalid Mode Transition Status Register (ME_IMTS) . . . 162
Figure 56. Debug Mode Transition Status Register (ME_DMTS) . . . 163
Figure 57. Invalid Mode Transition Status Register (ME_IMTS) . . . 165
Figure 58. TEST Mode Configuration Register (ME_TEST_MC) . . . 166
Figure 59. SAFE Mode Configuration Register (ME_SAFE_MC) . . . 166
Figure 60. DRUN Mode Configuration Register (ME_DRUN_MC) . . . 167
Figure 61. RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC) . . . 167
Figure 62. HALT Mode Configuration Register (ME_HALT_MC) . . . 168
Figure 63. STOP Mode Configuration Register (ME_STOP_MC) . . . 168
Figure 64. STANDBY Mode Configuration Register (ME_STANDBY_MC) . . . 169
Figure 65. Peripheral Status Register 0 (ME_PS0) . . . 171
Figure 66. Peripheral Status Register 1 (ME_PS1) . . . 171
Figure 67. Peripheral Status Register 2 (ME_PS2) . . . 172
Figure 68. Peripheral Status Register 3 (ME_PS3) . . . 172
Figure 69. Run Peripheral Configuration Registers (ME_RUN_PC0…7) . . . 173
Figure 70. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) . . . 174
Figure 71. Peripheral Control Registers (ME_PCTL0…143) . . . 175
Figure 72. MC_ME Mode Diagram . . . 177
Figure 73. MC_ME Transition Diagram . . . 190
Figure 74. MC_ME Application Example Flow Diagram . . . 194
Figure 75. MC_RGM block diagram . . . 196
Figure 76. Functional Event Status Register (RGM_FES) . . . 201
Figure 77. Destructive Event Status Register (RGM_DES) . . . 202
Figure 78. Functional Event Reset Disable Register (RGM_FERD) . . . 204
Figure 79. Destructive Event Reset Disable Register (RGM_DERD) . . . 205
Figure 80. Functional Event Alternate Request Register (RGM_FEAR) . . . 206
Figure 81. Destructive Event Alternate Request Register (RGM_DEAR) . . . 208
Figure 82. Functional Event Short Sequence Register (RGM_FESS) . . . 209
Figure 83. STANDBY Reset Sequence Register (RGM_STDBY) . . . 210
Figure 84. Functional Bidirectional Reset Enable Register (RGM_FBRE) . . . 211
Figure 85. MC_RGM State Machine . . . 214
Figure 86. MC_PCU Block Diagram . . . 220
Figure 87. Power Domain #0 Configuration Register (PCU_PCONF0) . . . 222
Figure 88. Power Domain #1 Configuration Register (PCU_PCONF1) . . . 224
Figure 89. Power Domain #2 Configuration Register (PCU_PCONF2) . . . 224
Figure 90. Power Domain Status Register (PCU_PSTAT) . . . 225
Figure 91. MC_PCU Events During Power Sequences (non-STANDBY mode) . . . 226
Figure 92. MC_PCU Events During Power Sequences (STANDBY mode) . . . 227
Figure 93. Voltage Regulator Control Register (VREG_CTL) . . . 231
Figure 94. Power domain organization . . . 233
Figure 95. WKPU block diagram . . . 235
Figure 96. NMI Status Flag Register (NSR) . . . 237
Figure 97. NMI Configuration Register (NCR) . . . 238
Figure 98. Wakeup/Interrupt Status Flag Register (WISR) . . . 239
Figure 99. Interrupt Request Enable Register (IRER) . . . 240
Figure 100. Wakeup Request Enable Register (WRER) . . . 240

Figure 101. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER) . . . 241
Figure 102. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER) . . . 241
Figure 103. Wakeup/Interrupt Filter Enable Register (WIFER) . . . 242
Figure 104. Wakeup/Interrupt Pullup Enable Register (WIPUER) . . . 242
Figure 105. NMI pad diagram . . . 243
Figure 106. External interrupt pad diagram . . . 245
Figure 107. RTC/API block diagram . . . 248
Figure 108. Clock gating for RTC clocks . . . 249
Figure 109. RTC Supervisor Control Register (RTCSUPV) . . . 250
Figure 110. RTC Control Register (RTCC) . . . 251
Figure 111. RTC Status Register (RTCS) . . . 253
Figure 112. RTC Counter Register (RTCCNT) . . . 254
Figure 113. Extended CAN data frame . . . 256
Figure 114. Control Register (CR) . . . 257
Figure 115. Sample register n . . . 258
Figure 116. e200z0h block diagram . . . 263
Figure 117. e200z0 SUPERVISOR Mode Program Model SPRs . . . 267
Figure 118. INTC block diagram . . . 270
Figure 119. INTC Module Configuration Register (INTC_MCR) . . . 273
Figure 120. INTC Current Priority Register (INTC_CPR) . . . 273
Figure 121. INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 0 . . . 275
Figure 122. INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 1 . . . 275
Figure 123. INTC End-of-Interrupt Register (INTC_EOIR) . . . 276
Figure 124. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3]) . . . 276
Figure 125. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7]) . . . 277
Figure 126. INTC Priority Select Register 0–3 (INTC_PSR[0:3]) . . . 278
Figure 127. INTC Priority Select Register 208-210 (INTC_PSR[208:210]) . . . 278
Figure 128. Software vector mode handshaking timing diagram . . . 290
Figure 129. Hardware vector mode handshaking timing diagram . . . 291
Figure 130. XBAR block diagram . . . 299
Figure 131. MPU block diagram . . . 305
Figure 132. MPU Control/Error Status Register (MPU_CESR) . . . 308
Figure 133. MPU Error Address Register, Slave Port n (MPU_EARn) . . . 309
Figure 134. MPU Error Detail Register, Slave Port n (MPU_EDRn) . . . 310
Figure 135. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0) . . . 311
Figure 136. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1) . . . 312
Figure 137. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2) . . . 313
Figure 138. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3) . . . 316
Figure 139. MPU RGD Alternate Access Control n (MPU_RGDAACn) . . . 317
Figure 140. MPU access evaluation macro . . . 320
Figure 141. System Integration Unit Lite block diagram . . . 324
Figure 142. MCU ID Register #1 (MIDR1) . . . 329
Figure 143. MCU ID Register #2 (MIDR2) . . . 330
Figure 144. Interrupt Status Flag Register (ISR) . . . 331
Figure 145. Interrupt Request Enable Register (IRER) . . . 332
Figure 146. Interrupt Rising-Edge Event Enable Register (IREER) . . . 332
Figure 147. Interrupt Falling-Edge Event Enable Register (IFEER) . . . 333
Figure 148. Interrupt Filter Enable Register (IFER) . . . 334
Figure 149. Pad Configuration Registers (PCRx) . . . 335
Figure 150. Pad Selection for Multiplexed Inputs Register (PSMI0_3) . . . 337
Figure 151. Port GPIO Pad Data Output Register 0–3 (GPDO0_3) . . . 340
Figure 152. Port GPIO Pad Data Input Register 0–3 (GPDI0_3) . . . 341

Figure 153. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC15) . . . 344
Figure 154. Interrupt Filter Clock Prescaler Register (IFCPR) . . . 345
Figure 155. Data Port example arrangement showing configuration for different port width accesses 346
Figure 156. External interrupt pad diagram . . . 347
Figure 157. I2C block diagram . . . 350
Figure 158. I2C Bus Address Register (IBAD) . . . 351
Figure 159. I2C Bus Frequency Divider Register (IBFD) . . . 352
Figure 160. SDA hold time . . . 353
Figure 161. SCL divider and SDA hold . . . 354
Figure 162. I2C Bus Control Register (IBCR) . . . 358
Figure 163. I2C Bus Status Register (IBSR) . . . 359
Figure 164. I2C Bus Data I/O Register (IBDR) . . . 360
Figure 165. I2C Bus Interrupt Config Register (IBIC) . . . 361
Figure 166. I2C bus transmission signals . . . 362
Figure 167. Start and stop conditions . . . 362
Figure 168. I2C bus clock synchronization . . . 364
Figure 169. Flow-Chart of Typical I2C Interrupt Routine . . . 370
Figure 170. LIN topology network . . . 373
Figure 171. LINFlex block diagram . . . 373
Figure 172. LINFlex operating modes . . . 375
Figure 173. LINFlex in loop back mode . . . 376
Figure 174. LINFlex in self test mode . . . 377
Figure 175. LIN control register 1 (LINCR1) . . . 378
Figure 176. LIN interrupt enable register (LINIER) . . . 381
Figure 177. LIN status register (LINSR) . . . 383
Figure 178. LIN error status register (LINESR) . . . 386
Figure 179. UART mode control register (UARTCR) . . . 387
Figure 180. UART mode status register (UARTSR) . . . 389
Figure 181. LIN timeout control status register (LINTCSR) . . . 391
Figure 182. LIN output compare register (LINOCR) . . . 392
Figure 183. LIN timeout control register (LINTOCR) . . . 392
Figure 184. LIN fractional baud rate register (LINFBRR) . . . 393
Figure 185. LIN integer baud rate register (LINIBRR) . . . 394
Figure 186. LIN checksum field register (LINCFR) . . . 395
Figure 187. LIN control register 2 (LINCR2) . . . 395
Figure 188. Buffer identifier register (BIDR) . . . 397
Figure 189. Buffer data register LSB (BDRL) . . . 398
Figure 190. Buffer data register MSB (BDRM) . . . 398
Figure 191. Identifier filter enable register (IFER) . . . 399
Figure 192. Identifier filter match index (IFMI) . . . 400
Figure 193. Identifier filter mode register (IFMR) . . . 401
Figure 194. Identifier filter control register (IFCR2*n*) . . . 402
Figure 195. Identifier filter control register (IFCR2*n* + 1) . . . 403
Figure 196. UART mode 8-bit data frame . . . 404
Figure 197. UART mode 9-bit data frame . . . 404
Figure 198. Filter configuration—register organization . . . 410
Figure 199. Identifier match index . . . 411
Figure 200. LIN synch field measurement . . . 412
Figure 201. Header and response timeout . . . 414
Figure 202. FlexCAN block diagram . . . 416
Figure 203. Message Buffer Structure . . . 421
Figure 204. Rx FIFO Structure . . . 425

Figure 205. ID Table 0 – 7 . . . 425
Figure 206. Module Configuration Register (MCR) . . . 427
Figure 207. Control Register (CTRL) . . . 431
Figure 208. Free Running Timer (TIMER) . . . 434
Figure 209. Rx Global Mask Register (RXGMASK) . . . 435
Figure 210. Error Counter Register (ECR) . . . 437
Figure 211. Error and Status Register (ESR) . . . 438
Figure 212. Interrupt Masks 2 Register (IMASK2) . . . 441
Figure 213. Interrupt Masks 1 Register (IMASK1) . . . 442
Figure 214. Interrupt Flags 2 Register (IFLAG2) . . . 443
Figure 215. Interrupt Flags 1 Register (IFLAG1) . . . 444
Figure 216. Rx Individual Mask Registers (RXIMR0 – RXIMR63) . . . 445
Figure 217. CAN engine clocking scheme . . . 454
Figure 218. Segments within the bit time . . . 455
Figure 219. Arbitration, match and move time windows . . . 456
Figure 220. DSPI block diagram . . . 461
Figure 221. DSPI with queues . . . 462
Figure 222. DSPI Module Configuration Register (DSPIx_MCR) . . . 467
Figure 223. DSPI Transfer Count Register (DSPIx_TCR) . . . 470
Figure 224. DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn) . . . 471
Figure 225. DSPI Status Register (DSPIx_SR) . . . 478
Figure 226. DSPI Interrupt Request Enable Register (DSPIx_RSER) . . . 480
Figure 227. DSPI PUSH TX FIFO Register (DSPIx_PUSHR) . . . 482
Figure 228. DSPI POP RX FIFO Register (DSPIx_POPR) . . . 484
Figure 229. DSPI Transmit FIFO Register 0–3 (DSPIx_TXFRn) . . . 485
Figure 230. DSPI Receive FIFO Registers 0–3 (DSPIx_RXFRn) . . . 486
Figure 231. SPI serial protocol overview . . . 487
Figure 232. DSPI start and stop state diagram . . . 489
Figure 233. Communications clock prescalers and scalers . . . 492
Figure 234. Peripheral chip select strobe timing . . . 494
Figure 235. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8) . . . 497
Figure 236. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8) . . . 498
Figure 237. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$) . . . 500
Figure 238. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$) . . . 501
Figure 239. Example of non-continuous format (CPHA = 1, CONT = 0) . . . 502
Figure 240. Example of continuous transfer (CPHA = 1, CONT = 1) . . . 502
Figure 241. Polarity switching between frames . . . 503
Figure 242. Continuous SCK timing diagram (CONT= 0) . . . 504
Figure 243. Continuous SCK timing diagram (CONT=1) . . . 505
Figure 244. TX FIFO pointers and counter . . . 511
Figure 245. Interaction between timers and relevant peripherals . . . 514
Figure 246. STM Control Register (STM_CR) . . . 519
Figure 247. STM Count Register (STM_CNT) . . . 519
Figure 248. STM Channel Control Register (STM_CCRn) . . . 520
Figure 249. STM Channel Interrupt Register (STM_CIRn) . . . 521
Figure 250. STM Channel Compare Register (STM_CMPn) . . . 521
Figure 251. Channel configuration . . . 524
Figure 252. eMIOS Module Configuration Register (EMIOSMCR) . . . 526
Figure 253. eMIOS Global FLAG (EMIOSGFLAG) Register . . . 528
Figure 254. eMIOS Output Update Disable (EMIOSOUDIS) Register . . . 528
Figure 255. eMIOS Enable Channel (EMIOSUCDIS) Register . . . 529
Figure 256. eMIOS UC A Register (EMIOSA[n]) . . . 529

Figure 257. eMIOS UC B Register (EMIOSB[n]) . . . 530
Figure 258. eMIOS UC Counter Register (EMIOSCNT[n]) . . . 531
Figure 259. eMIOS UC Control Register (EMIOSC[n]) . . . 532
Figure 260. eMIOS UC Status Register (EMIOSS[n]) . . . 536
Figure 261. eMIOS UC Alternate A register (EMIOSALTA[n]) . . . 537
Figure 262. Single action input capture with rising edge triggering example . . . 539
Figure 263. Single action input capture with both edges triggering example . . . 539
Figure 264. SAOC example with EDPOL value being transferred to the output flip-flop . . . 540
Figure 265. SAOC example toggling the output flip-flop . . . 540
Figure 266. SAOC example with flag behavior . . . 541
Figure 267. Input pulse width measurement example . . . 542
Figure 268. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads . . . 542
Figure 269. Input period measurement example . . . 543
Figure 270. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads . . . 544
Figure 271. Double action output compare with FLAG set on the second match . . . 545
Figure 272. Double action output compare with FLAG set on both matches . . . 545
Figure 273. DAOC with transfer disabling example . . . 546
Figure 274. Modulus Counter Up mode example . . . 547
Figure 275. Modulus Counter Up/Down mode example . . . 548
Figure 276. Modulus Counter Buffered (MCB) Up Count mode . . . 549
Figure 277. Modulus Counter Buffered (MCB) Up/Down mode . . . 549
Figure 278. MCB Mode A1 Register Update in Up Counter mode . . . 550
Figure 279. MCB Mode A1 Register Update in Up/Down Counter mode . . . 550
Figure 280. OPWFMB A1 and B1 match to Output Register Delay . . . 551
Figure 281. OPWFMB Mode with A1 = 0 (0% duty cycle) . . . 552
Figure 282. OPWFMB A1 and B1 registers update and flags . . . 553
Figure 283. OPWFMB mode from 100% to 0% duty cycle . . . 553
Figure 284. OPWMCB A1 and B1 registers load . . . 555
Figure 285. OPWMCB with lead dead time insertion . . . 556
Figure 286. OPWMCB with trail dead time insertion . . . 557
Figure 287. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3) . . . 559
Figure 288. OPWMB mode matches and flags . . . 560
Figure 289. OPWMB mode with 0% duty cycle . . . 561
Figure 290. OPWMB mode from 100% to 0% duty cycle . . . 561
Figure 291. OPWMT example . . . 564
Figure 292. OPWMT with 0% Duty Cycle . . . 564
Figure 293. OPWMT with 100% duty cycle . . . 565
Figure 294. Input programmable filter submodule diagram . . . 565
Figure 295. Input programmable filter example . . . 566
Figure 296. Time base period when running in the fastest prescaler ratio . . . 568
Figure 297. Time base generation with external clock and clear on match start . . . 569
Figure 298. Time base generation with internal clock and clear on match start . . . 569
Figure 299. Time base generation with clear on match end . . . 570
Figure 300. PIT block diagram . . . 571
Figure 301. PIT Module Control Register (PITMCR) . . . 573
Figure 302. Timer Load Value Register (LDVAL) . . . 573
Figure 303. Current Timer Value Register (CVAL) . . . 574
Figure 304. Timer Control Register (TCTRL) . . . 575
Figure 305. Timer Flag Register (TFLG) . . . 575
Figure 306. Stopping and starting a timer . . . 576
Figure 307. Modifying running timer period . . . 577
Figure 308. Dynamically setting a new load value . . . 577

Figure 309. ADC implementation . . . 580
Figure 310. Normal conversion flow . . . 582
Figure 311. Injected sample/conversion sequence . . . 583
Figure 312. Sampling and conversion timings . . . 585
Figure 313. Presampling sequence . . . 588
Figure 314. Presampling sequence with PRECONV = 1 . . . 588
Figure 315. Guarded area . . . 589
Figure 316. Main Configuration Register (MCR) . . . 597
Figure 317. Main Status Register (MSR) . . . 599
Figure 318. Interrupt Status Register (ISR) . . . 600
Figure 319. Channel Pending Register 0 (CEOCFR0) . . . 601
Figure 320. Channel Pending Register 1 (CEOCFR1) . . . 601
Figure 321. Channel Pending Register 2 (CEOCFR2) . . . 602
Figure 322. Interrupt Mask Register (IMR) . . . 602
Figure 323. Channel Interrupt Mask Register 0 (CIMR0) . . . 603
Figure 324. Channel Interrupt Mask Register 1 (CIMR1) . . . 604
Figure 325. Channel Interrupt Mask Register 2 (CIMR2) . . . 604
Figure 326. Watchdog Threshold Interrupt Status Register (WTISR) . . . 605
Figure 327. Watchdog Threshold Interrupt Mask Register (WTIMR) . . . 605
Figure 328. Threshold Control Register (TRCx, x = [0..3]) . . . 607
Figure 329. Threshold Register (THRHLR[0:3]) . . . 608
Figure 330. Presampling Control Register (PSCR) . . . 608
Figure 331. Presampling Register 0 (PSR0) . . . 609
Figure 332. Presampling Register 1 (PSR1) . . . 610
Figure 333. Presampling Register 2 (PSR2) . . . 610
Figure 334. Conversion timing registers CTR[0..2] . . . 611
Figure 335. Normal Conversion Mask Register 0 (NCMR0) . . . 612
Figure 336. Normal Conversion Mask Register 1 (NCMR1) . . . 612
Figure 337. Normal Conversion Mask Register 2 (NCMR2) . . . 613
Figure 338. Injected Conversion Mask Register 0 (JCMR0) . . . 614
Figure 339. Injected Conversion Mask Register 1 (JCMR1) . . . 614
Figure 340. Injected Conversion Mask Register 2 (JCMR2) . . . 615
Figure 341. Decode Signals Delay Register (DSDR) . . . 616
Figure 342. Power-down Exit Delay Register (PDEDR) . . . 616
Figure 343. Channel Data Register (CDR[0..95]) . . . 617
Figure 344. Cross Triggering Unit block diagram . . . 619
Figure 345. Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63) . . . 620
Figure 346. Flash memory architecture . . . 629
Figure 347. CFlash and DFlash module structures . . . 631
Figure 348. CFlash Module Configuration Register (CFLASH_MCR) . . . 639
Figure 349. CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML) . . . 644
Figure 350. CFlash Nonvolatile Low/Mid address space block Locking register (CFLASH_NVLML) . 647
Figure 351. CFlash Secondary Low/mid address space block Locking Register (CFLASH_SLL) . . . 649
Figure 352. CFlash Nonvolatile Secondary Low/mid address space block Locking register (CFLASH_NVSLL) . . . 651
Figure 353. CFlash Low/Mid address space block Select register (CFLASH_LMS) . . . 654
Figure 354. CFlash Address Register (CFLASH_ADR) . . . 655
Figure 355. CFlash User Test 0 register (CFLASH_UT0) . . . 657
Figure 356. CFlash User Test 1 register (CFLASH_UT1) . . . 659
Figure 357. CFlash User Test 2 register (CFLASH_UT2) . . . 659
Figure 358. CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0) . . . 660
Figure 359. CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1) . . . 661

Figure 360. CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2). 662
Figure 361. CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3). 663
Figure 362. CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4). 664
Figure 363. CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0) 665
Figure 364. CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1) 665
Figure 365. CFlash Nonvolatile System Censorship Control 0 register (NVSCC0). 666
Figure 366. CFlash Nonvolatile System Censorship Control 1 register (NVSCC1). 667
Figure 367. CFlash Nonvolatile User Options register (NVUSRO) 668
Figure 368. DFlash Module Configuration Register (DFLASH_MCR) 669
Figure 369. DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML). 674
Figure 370. DFlash Nonvolatile Low/Mid address space block Locking register (DFLASH_NVLML) . 676
Figure 371. DFlash Secondary Low/mid address space block Locking register (DFLASH_SLL) 678
Figure 372. DFlash Nonvolatile Secondary Low/mid address space block Locking register (DFLASH_NVSLL) 680
Figure 373. DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS). 682
Figure 374. DFlash Address Register (DFLASH_ADR). 683
Figure 375. DFlash User Test 0 register (DFLASH_UT0) 684
Figure 376. DFlash User Test 1 register (DFLASH_UT1) 686
Figure 377. DFlash User Test 2 register (DFLASH_UT2) 687
Figure 378. DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0). 688
Figure 379. DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1). 689
Figure 380. DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2). 690
Figure 381. DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3). 691
Figure 382. DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4). 692
Figure 383. Power Architecture e200z0h RPP reference platform block diagram. 704
Figure 384. PFlash Configuration Register 0 (PFCR0) 708
Figure 385. PFlash Configuration Register 1 (PFCR1) 712
Figure 386. PFlash Access Protection Register (PFAPR). 714
Figure 387. Nonvolatile Platform Flash Access Protection Register (NVPFAPR) 715
Figure 388. Register Protection block diagram 723
Figure 389. Register protection memory diagram 724
Figure 390. Soft Lock Bit Register (SLBR*n*) 726
Figure 391. Global Configuration Register (GCR) 727
Figure 392. Change Lock Settings Directly Via Area #4 729
Figure 393. Change Lock Settings for 16-bit Protected Addresses 729
Figure 394. Change Lock Settings for 32-bit Protected Addresses 730
Figure 395. Change Lock Settings for Mixed Protection 730
Figure 396. Enable Locking Via Mirror Module Space (Area #3). 731
Figure 397. Enable Locking for Protected and Unprotected Addresses 731
Figure 398. SWT Control Register (SWT_CR) 739
Figure 399. SWT Interrupt Register (SWT_IR) 740
Figure 400. SWT Time-Out Register (SWT_TO). 741
Figure 401. SWT Window Register (SWT_WN) 741
Figure 402. SWT Service Register (SWT_SR) 742
Figure 403. SWT Counter Output Register (SWT_CO). 742
Figure 404. Processor Core Type Register (PCT). 747
Figure 405. SoC-Defined Platform Revision Register (REV). 747
Figure 406. IPS On-Platform Module Configuration Register (IOPMC). 748
Figure 407. Miscellaneous Wakeup Control (MWCR) Register. 749
Figure 408. Miscellaneous Interrupt (MIR) Register 750
Figure 409. Miscellaneous User-Defined Control (MUDCR) Register. 751
Figure 410. ECC Configuration (ECR) Register 752

Figure 411. ECC Status Register (ESR) . . . 755
Figure 412. ECC Error Generation Register (EEGR) . . . 756
Figure 413. Platform Flash ECC Address Register (PFEAR) . . . 759
Figure 414. Platform Flash ECC Master Number Register (PFEMR) . . . 760
Figure 415. Platform Flash ECC Attributes Register (PFEAT) . . . 760
Figure 416. Platform Flash ECC Data Register (PFEDR) . . . 761
Figure 417. Platform RAM ECC Address Register (PREAR) . . . 762
Figure 418. Platform RAM ECC Syndrome Register (PRESR) . . . 763
Figure 419. Platform RAM ECC Master Number Register (PREMR) . . . 765
Figure 420. Platform RAM ECC Attributes Register (PREAT) . . . 765
Figure 421. Platform RAM ECC Data Register (PREDR) . . . 766
Figure 422. JTAG Controller Block Diagram . . . 768
Figure 423. 5-bit Instruction Register . . . 771
Figure 424. Device Identification Register . . . 771
Figure 425. Shifting data through a register . . . 772
Figure 426. IEEE 1149.1-2001 TAP controller finite state machine . . . 773
Figure 427. e200z0 OnCE Block Diagram . . . 777
Figure 428. OnCE Command Register (OCMD) . . . 778
Figure 429. NDI Functional Block Diagram . . . 780
Figure 430. NDI Implementation Block Diagram . . . 781
Figure 431. Nexus Device ID (DID) Register . . . 785
Figure 432. Port Configuration Register (PCR) . . . 786
Figure 433. Development Control Register 1 (DC1) . . . 788
Figure 434. Development Control Register 2 (DC2) . . . 789
Figure 435. Development Status (DS) Register . . . 790
Figure 436. Read/Write Access Control/Status (RWCS) Register . . . 792
Figure 437. Read/Write Access Address (RWA) Register . . . 793
Figure 438. Read/Write Access Data (RWD) Register . . . 794
Figure 439. Watchpoint Trigger (WT) Register . . . 794
Figure 440. Ownership Trace Message Format . . . 798
Figure 441. Error Message Format . . . 799

1 Preface

1.1 Overview

The primary objective of this document is to define the functionality of the SPC560Bx and SPC560Cx microcontroller for use by software and hardware developers. The SPC560Bx and SPC560Cx is built on Power Architecture® technology and integrates technologies that are important for today's automotive vehicle body applications.

The information in this book is subject to change without notice, as described in the disclaimers on the title page. 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.

1.2 Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC560Bx and SPC560Cx 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.

1.3 Guide to this reference manual

Table 1. Guide to this reference manual

Chapter		Description	Functional group
#	Title		
2	*Introduction*	General overview, family description, feature list and information on how to use the reference manual in conjunction with other available documents.	Introductory material
3	*Memory Map*	Memory map of all peripherals and memory.	Memory map
4	*Signal description*	Pinout diagrams and descriptions of all pads.	Signals
5	*Microcontroller Boot*		Boot
	– *Boot mechanism*	– Describes what configuration is required by the user and what processes are involved when the microcontroller boots from flash memory or serial boot modes. – Describes censorship.	
	– *Boot Assist Module (BAM)*	Features of BAM code and when it's used.	
	– *System Status and Configuration Module (SSCM)*	Reports information about current state and configuration of the microcontroller.	

Table 1. Guide to this reference manual (continued)

Chapter		Description	Functional group
#	Title		
6	*Clock Description*	– Covers configuration of all of the clock sources in the system. – Describes the Clock Monitor Unit (CMU).	Clocks and power (includes operating mode configuration and how to wake up from low power mode)
7	*Clock Generation Module (MC_CGM)*	Determines how the clock sources are used (including clock dividers) to generate the reference clocks for all of the modules and peripherals.	
8	*Mode Entry Module (MC_ME)*	Determines the clock source, memory, power and peripherals that are available in each operating mode.	
9	*Reset Generation Module (MC_RGM)*	Manages the process of entering and exiting reset, allows reset sources to be configured (including LVD's) and provides status reporting.	
10	*Power Control Unit (MC_PCU)*	Controls the power to different power domains within the microcontroller (allowing SRAM to be selectively powered in STANDBY mode).	
11	*Voltage Regulators and Power Supplies*	Information on voltage regulator implementation. Includes enable bit for 5 V LVD (see also MC_RGM).	
12	*Wakeup Unit (WKPU)*	Always-active analog block. Details configuration of 2 internal (API/RTC) and 30 external (pin) low power mode wakeup sources.	
13	*Real Time Clock / Autonomous Periodic Interrupt (RTC/API)*	Details configuration and operation of timers that are predominately used for system wakeup.	
14	*CAN Sampler*	Details on how to configure the CAN sampler which is used to capture the identifier frame of a CAN message when the microcontroller is in low power mode.	
15	*e200z0h Core*	Overview on cores. For more details consult the core reference manuals available on www.st.com.	Core platform modules
16	*Interrupt Controller (INTC)*	Provides the configuration and control of all of the external interrupts (non-core) that are then routed to the IVOR4 core interrupt vector.	
17	*Crossbar Switch (XBAR)*	Describes the connections of the XBAR masters and slaves on this microcontroller.	
18	*Memory Protection Unit (MPU)*	The MPU sits on the slave side of the XBAR and allows highly configurable control over all master accesses to the memory.	
19	*System Integration Unit Lite (SIUL)*	How to configure the pins or ports for input or output functions including external interrupts and DSI serialization.	Ports

Table 1. Guide to this reference manual (continued)

Chapter		Description	Functional group
#	Title		
20	*Inter-Integrated Circuit Bus Controller Module (I^2C)*	These chapters describe the configuration and operation of the various communication modules. Some of these modules support eDMA requests to fill / empty buffer queues to minimize CPU overhead.	Communication modules
21	*LIN Controller (LINFlex)*		
22	*FlexCAN*		
23	*Deserial Serial Peripheral Interface (DSPI)*		
24	*Timers*		Timer modules
	– *Technical overview*	Gives an overview of the available system timer modules showing links to other modules as well as tables detailing the external pins associated with eMIOS timer channels.	
	– *System Timer Module (STM)*	A simple 32-bit free running counter with 4 compare channels with interrupt on match. It can be read at any time; this is very useful for measuring execution times.	
	– *Enhanced Modular IO Subsystem (eMIOS)*	Highly configurable timer module(s) supporting PWM, output compare and input capture features. Includes interrupt and eDMA support.	
	– *Periodic Interrupt Timer (PIT)*	Set of 32-bit countdown timers that provide periodic events (which can trigger an interrupt) with automatic re-load.	
25	*Analog-to-Digital Converter (ADC)*	Details the configuration and operation of the ADC modules as well as detailing the channels that are shared between the 10-bit and 12-bit ADC. The ADC is tightly linked to the INTC, eDMA, PIT_RTI and CTU. When used in conjunction with these other modules, the CPU overhead for an ADC conversion is significantly reduced.	ADC system
26	*Cross Triggering Unit (CTU)*	The CTU allows an ADC conversion to be automatically triggered based on an eMIOS event (like a PWM output going high) or a PIT_RTI event with no CPU intervention.	
28	*Flash Memory*	Details the code and data flash memory structure (with ECC), block sizes and the flash memory port configuration, including wait states, line buffer configuration and pre-fetch control.	Memory
27	*Static RAM (SRAM)*	Details the structure of the SRAM (with ECC). There are no user configurable registers associated with the SRAM.	

Table 1. Guide to this reference manual (continued)

Chapter		Description	Functional group
#	Title		
29	*Register Protection*	Certain registers in each peripheral can be protected from further writes using the register protection mechanism detailed in this section. Registers can either be configured to be unlocked via a soft lock bit or locked unit the next reset.	Integrity
30	*Software Watchdog Timer (SWT)*	The SWT offers a selection of configurable modes that can be used to monitor the operation of the microcontroller and /or reset the device or trigger an interrupt if the SWT is not correctly serviced. The SWT is enabled out of reset.	
31	*Error Correction Status Module (ECSM)*	Provides information about the last reset, general device information, system fault information and detailed ECC error information.	
32	*IEEE 1149.1 Test Access Port Controller (JTAGC)*	Used for boundary scan as well as device debug.	Debug
33	*Nexus Development Interface (NDI)*	Provides advanced debug features including non intrusive trace capabilities.	
A	*Register Map*	Summarizes the registers on this microcontroller	Register summary
	Revision History	Summarizes the changes between each successive revision of this reference manual	Revision history information

1.4 Register description conventions

The register information for SPC560Bx and SPC560Cx is presented in:

- Memory maps containing:
  - An offset from the module's base address
  - The name and acronym/abbreviation of each register
  - The page number on which each register is described
- Register figures
- Field-description tables
- Associated text

The register figures show the field structure using the conventions in *Figure 1*.

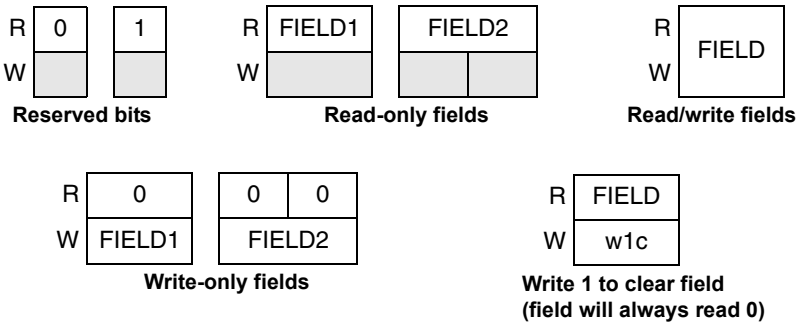


Figure 1. Register figure conventions

The numbering of register bits and fields on SPC560Bx and SPC560Cx is as follows:

- Register bit numbers, shown at the top of each figure, use the standard Power Architecture bit ordering (0, 1, 2, ...) where bit 0 is the most significant bit (MSB).
- Multi-bit fields within a register use conventional bit ordering (..., 2, 1, 0) where bit 0 is the least significant bit (LSB).

1.5 References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC560Bx and SPC560Cx:

- IEEE-ISTO 5001-2003 Standard for a Global Embedded Processor Interface (Nexus)
- IEEE 1149.1-2001 standard - IEEE Standard Test Access Port and Boundary-Scan Architecture

1.6 How to use the SPC560Bx and SPC560Cx documents

This section:

- Describes how the SPC560Bx and SPC560Cx documents provide information on the microcontroller
- Makes recommendations on how to use the documents in a system design

1.6.1 The SPC560Bx and SPC560Cx document set

The SPC560Bx and SPC560Cx document set comprises:

- This reference manual (provides information on the features of the logical blocks on the device and how they are integrated with each other)
- The device data sheet (specifies the electrical characteristics of the device)
- The device product brief

The following reference documents (available online at www.st.com) are also available to support the CPU on this device:

- *Programmer's Reference Manual for Book E Processors*
- *Variable-Length Encoding (VLE) Extension - Programming Interface Manual*

The aforementioned documents describe all of the functional and electrical characteristics of the SPC560Bx and SPC560Cx microcontroller.

Depending on your task, you may need to refer to multiple documents to make design decisions. However, in general the use of the documents can be divided up as follows:

- Use the reference manual (this document) during software development and when allocating functions during system design.
- Use the data sheet when designing hardware and optimizing power consumption.
- Use the CPU reference documents when doing detailed software development in assembly language or debugging complex software interactions.

1.6.2 Reference manual content

The content in this document focuses on the functionality of the microcontroller rather than its performance. Most chapters describe the functionality of a particular on-chip module, such as a CAN controller or timer. The remaining chapters describe how these modules are integrated into the memory map, how they are powered and clocked, and the pin-out of the device.

In general, when an individual module is enabled for use all of the detail required to configure and operate it is contained in the dedicated chapter. In some cases there are multiple implementations of this module, however, there is only one chapter for each type of module in use. For this reason, the address of registers in each module is normally provided as an offset from a base address which can be found in *3, Memory Map*. The benefit of this approach is that software developed for a particular module can be easily reused on this device and on other related devices that use the same modules.

The steps to enable a module for use varies but typically these require configuration of the integration features of the microcontroller. The module will normally have to be powered and enabled at system level, then a clock may have to be explicitly chosen and finally if required the input and output connections to the external system must be configured.

The primary integration chapters of the reference manual contain most of the information required to enable the modules. There are special cases where a chapter may describe module functionality and some integration features for convenience — for example, the microcontroller input/output (SIUL) module. Integration and functional content is provided in the manual as shown in *Table 2*.

Table 2. Reference manual integration and functional content

Chapter	Integration content	Functional content
Introduction	– The main features on chip – A summary of the functions provided by each module	—
Memory Map	How the memory map is allocated, including: – Internal RAM – Flash memory – External memory-mapped resources and the location of the registers used by the peripherals(1)	—

Table 2. Reference manual integration and functional content (continued)

Chapter	Integration content	Functional content
Signal Description	How the signals from each of the modules are combined and brought to a particular pin on a package	—
Boot Assist Module	CPU boot sequence from reset	Implementation of the boot options if internal flash memory is not used
Clock Description	Clocking architecture of the device (which clock is available for the system and each peripheral)	Description of operation of different clock sources
Interrupt Controller	Interrupt vector table	Operation of the module
Mode Entry Module	Module numbering for control and status	Operation of operating modes
System Integration Unit Lite	How input signals are mapped to individual modules including external interrupt pins	Operation of GPIO
Voltage regulators and power supplies	Power distribution to the MCU	—
Wakeup Unit	Allocation of inputs to the Wakeup Unit	Operation of the wakeup feature

1. To find the address of a register in a particular module take the start address of the module given in the memory map and add the offset for the register given in the module chapter.

1.7 Using the SPC560Bx and SPC560Cx

There are many different approaches to designing a system using the SPC560Bx and SPC560Cx so the guidance in this section is provided as an example of how the documents can be applied in this task.

Familiarity with the SPC560Bx and SPC560Cx modules can help ensure that its features are being optimally used in a system design. Therefore, the current chapter is a good starting point. Further information on the detailed features of a module are provided within the module chapters. These, combined with the current chapter, should provide a good introduction to the functions available on the MCU.

1.7.1 Hardware design

The SPC560Bx and SPC560Cx requires that certain pins are connected to particular power supplies, system functions and other voltage levels for operation.

The SPC560Bx and SPC560Cx internal logic operates from 1.2 V (nominal) supplies that are normally supplied by the on-chip voltage regulator from a 5 V or 3.3 V supply. The 3.3–5 V (±10%) supply is also used to supply the input/output pins on the MCU. *4, Signal description,* describes the power supply pin names, numbers and their purpose. For more detail on the voltage supply of each pin, see *11, Voltage Regulators and Power Supplies*. For specifications of the voltage ranges and limits and decoupling of the power supplies see the SPC560Bx and SPC560Cx data sheet.

Certain pins have dedicated functions that affect the behavior of the MCU after reset. These include pins to force test or alternate boot conditions and debug features. These are described in *4, Signal description,* and a hardware designer should take care that these pins are connected to allow correct operation.

Beyond power supply and pins that have special functions there are also pins that have special system purposes such as oscillator and reset pins. These are also described in *4, Signal description*. The reset pin is bidirectional and its function is closely tied to the reset generation module [*9, Reset Generation Module (MC_RGM)*]. The crystal oscillator pins are dedicated to this function but the oscillator is not started automatically after reset. The oscillator module is described in *6, Clock Description,* along with the internal clock architecture and the other oscillator sources on chip.

1.7.2 Input/output pins

The majority of the pins on the MCU are input/output pins which may either operate as general purpose pins or be connected to a particular on-chip module. The arrangement allows a function to be available on several pins. The system designer should allocate the function for the pin before connecting to external hardware. The software should then choose the correct function to match the hardware. The pad characteristics can vary depending on the functions on the pad. *4, Signal description,* describes each pad type (for example, S, M, or J). Two pads may be able to carry the same function but have different pad types. The electrical specification of the pads is described in the data sheet dependent on the function enabled and the pad type.

There are three modules that configure the various functions available:

- System Integration Unit Lite (SIUL)
- Wakeup Unit (WKPU)
- 32 KHz oscillator (SXOSC)

The SIUL configures the digital pin functions. Each pin has a register (PCR) in the module that allows selection of the output functions that is connected to the pin. The available settings for the PCR are described in *Section 4.7, Functional ports.* Inputs are selected using the PSMI registers; these are described in *19, System Integration Unit Lite (SIUL)*. (PSMI registers connect a module to one of several pins, whereas the PCR registers connect a pin to one of several modules).

The WKPU provides the ability to cause interrupts and wake the MCU from low power modes and operates independently from the SIUL.

In addition to digital I/O functions the SXOSC is a "special function" that provides a slow external crystal. The SXOSC is enabled independently from the digital I/O which means that the digital function on the pin must be disabled when the SXOSC is active. The ADC functions are enabled using the PCRs.

1.7.3 Software design

Certain modules provide system integration functions, and other modules (such as timers) provide specific functions.

From reset, the modules involved in configuring the system for application software are:

- Boot Assist Module (BAM) — determines the selected boot source
- Reset Generation Module (MC_RGM) — determines the behavior of the MCU when various reset sources are triggered and reports the source of the reset
- Mode Entry Module (MC_ME) — controls which operating mode the MCU is in and configures the peripherals and clocks and power supplies for each of the modes
- Power Control Unit (MC_PCU) — determines which power domains are active
- Clock Generation Module (MC_CGM) — chooses the clock source for the system and many peripherals

After reset, the MCU will automatically select the appropriate reset source and begin to execute code. At this point the system clock is the 16 MHz FIRC oscillator, the CPU is in supervisor mode and all the memory is available. Initialization is required before most peripherals may be used and before the SRAM can be read (since the SRAM is protected by ECC, the syndrome will generally be uninitialized after reset and reads would fail the check). Accessing disabled features causes error conditions or interrupts.

A typical startup routine would involve initializing the software environment including stacks, heaps, variable initialization and so on and configuring the MCU for the application.

The MC_ME module enables the modules and other features like clocks. It is therefore an essential part of the initialization and operation software. In general, the software will configure an MC_ME mode to make certain peripherals, clocks, and memory active and then switch to that mode.

6, Clock Description, includes a graphic of the clock architecture of the MCU. This can be used to determine how to configure the MC_CGM module. In general software will configure the module to enable the required clocks and PLLs and route these to the active modules.

After these steps are complete it is possible to configure the input/output pins and the modules for the application.

1.7.4 Other features

The MC_ME module manages low power modes and so it is likely that it will be used to switch into different configurations (module sets, clocks) depending on the application requirements.

The MCU includes two other features to improve the integrity of the application:

- It is possible to enable a software watchdog (SWT) immediately at reset or afterwards to help detect code runaway.
- Individual register settings can be protected from unintended writes using the features of the Register Protection module. The protected registers are shown in *29, Register Protection*.

Other integration functionality is provided by the System Status and Configuration Module (SSCM).

2 Introduction

2.1 The SPC560Bx and SPC560Cx microcontroller family

The SPC560Bx and SPC560Cx represents a new generation of 32-bit microcontrollers based on the Power Architecture®. It belongs to an expanding family of automotive-focused products targeted at addressing the next wave of body electronics applications within the vehicle.

This document describes the features of the family and options available within the family members, and highlights important electrical and physical characteristics of the device.

The advanced and cost-efficient host processor core of the family complies with the Power Architecture embedded category. It operates at speeds of up to 64 MHz and offers high performance processing optimized for low power consumption. It capitalizes on the available development infrastructure of current Power Architecture devices and is supported with software drivers, operating systems and configuration code to assist with users implementations. See *Section 2.4, Developer support*, for more information.

2.2 Features

This section describes the features of the SPC560Bx and SPC560Cx.

2.2.1 SPC560Bx and SPC560Cx family comparison

Table 3 and *Table 4* report the memory scaling of Code Flash and SRAM.

Table 3. Code Flash memory scaling

Memory size	Start address	End address
256 KB	0x00000000	0x0003FFFF
384 KB	0x00000000	0x0005FFFF
512 KB	0x00000000	0x0007FFFF

Table 4. SRAM memory scaling

Memory size	Start address	End address
24 KB	0x40000000	0x40005FFF
28 KB	0x40000000	0x40006FFF
32 KB	0x40000000	0x40007FFF
40 KB	0x40000000	0x40009FFF
48 KB	0x40000000	0x4000BFFF

Table 5 provides a summary of the different members of the SPC560Bx and SPC560Cx family. This information is intended to provide an understanding of the range of functionality offered by this family.

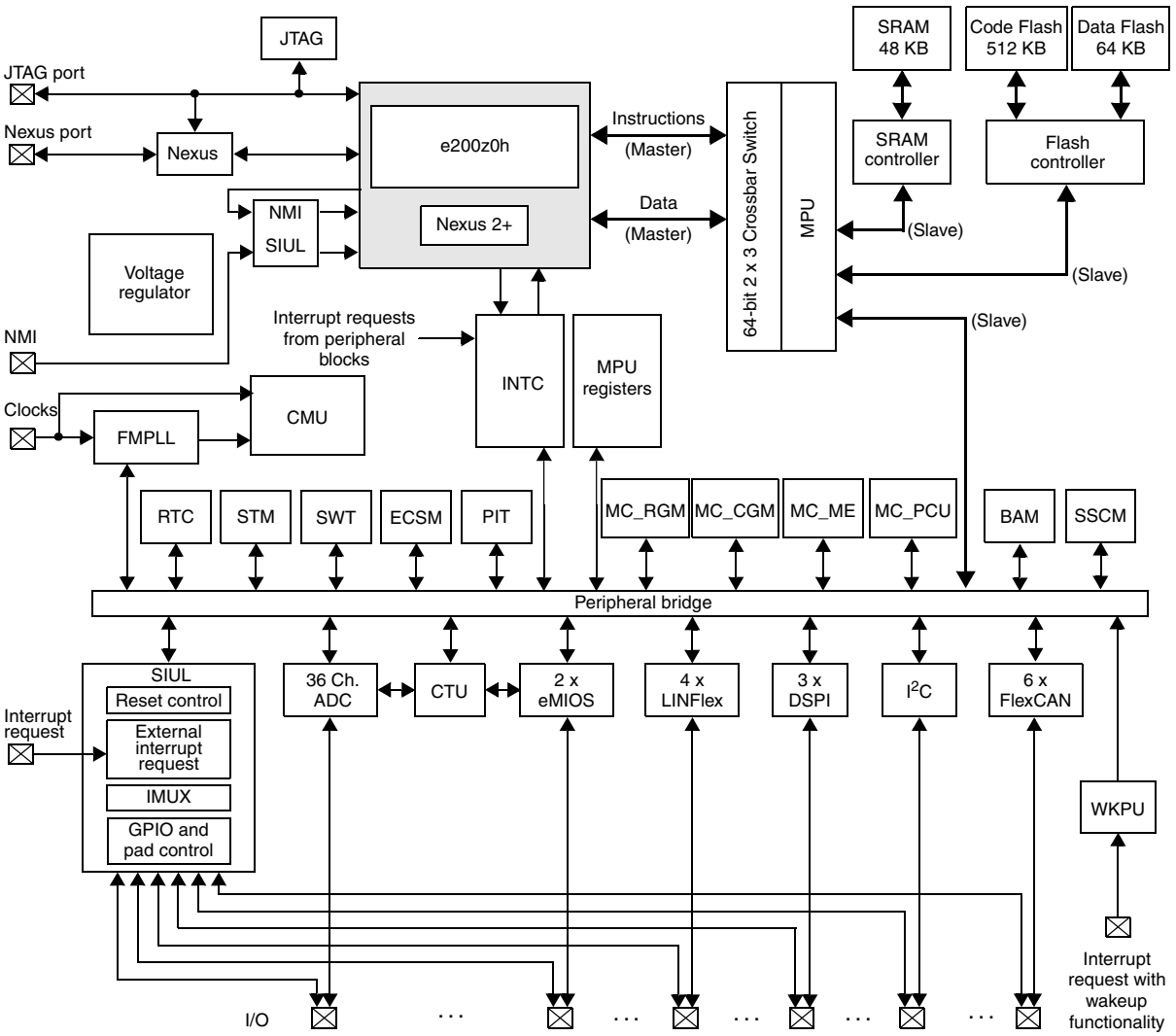
Table 5. SPC560Bx and SPC560Cx device comparison[1]

Feature	Device										
	SPC560B40L1	**SPC560B40L3**	**SPC560B40L5**	**SPC560C40L1**	**SPC560C40L3**	**SPC560B50L1**	**SPC560B50L3**	**SPC560B50L5**	**SPC560C50L1**	**SPC560C50L3**	**SPC560B50B2**
CPU	e200z0h										
Execution speed[2]	Static – up to 64 MHz										
Code Flash	256 KB					512 KB					
Data Flash	64 KB (4 × 16 KB)										
RAM	24 KB			32 KB		32 KB			48 KB		
MPU	8-entry										
ADC	12 ch, 10-bit	28 ch, 10-bit	36 ch, 10-bit	8 ch, 10-bit	28 ch, 10-bit	12 ch, 10-bit	28 ch, 10-bit	36 ch, 10-bit	8 ch, 10-bit	28 ch, 10-bit	36 ch, 10-bit
CTU	Yes										
Total timer I/O[3] eMIOS	12 ch, 16-bit	28 ch, 16-bit	56 ch, 16-bit	12 ch, 16-bit	28 ch, 16-bit	12 ch, 16-bit	28 ch, 16-bit	56 ch, 16-bit	12 ch, 16-bit	28 ch, 16-bit	56 ch, 16-bit
– PWM + MC + IC/OC[4]	2 ch	5 ch	10 ch	2 ch	5 ch	2 ch	5 ch	10 ch	2 ch	5 ch	10 ch
– PWM + IC/OC[4]	10 ch	20 ch	40 ch	10 ch	20 ch	10 ch	20 ch	40 ch	10 ch	20 ch	40 ch
– IC/OC[4]	0 ch	3 ch	6 ch	0 ch	3 ch	0 ch	3 ch	6 ch	0 ch	3 ch	6 ch
SCI (LINFlex)	3[5]			4							
SPI (DSPI)	2	3		2	3	2	3		2	3	
CAN (FlexCAN)	2[6]			5	6	3[7]			5	6	
I^2C	1										
32 kHz oscillator	Yes										
GPIO[7]	45	79	123	45	79	45	79	123	45	79	123
Debug	JTAG										Nexus2+
Package	LQFP 64[8]	LQFP 100	LQFP 144	LQFP 64[8]	LQFP 100	LQFP 64[8]	LQFP 100	LQFP 144	LQFP 64[8]	LQFP 100	LBGA 208[9]

1. Feature set dependent on selected peripheral multiplexing—table shows example implementation
2. Based on 125 °C ambient operating temperature
3. See the eMIOS section of the device reference manual for information on the channel configuration and functions.
4. IC - Input Capture; OC - Output Compare; PWM - Pulse Width Modulation; MC - Modulus counter
5. SCI0, SCI1 and SCI2 are available. SCI3 is not available.
6. CAN0, CAN1 are available. CAN2, CAN3, CAN4 and CAN5 are not available.
7. I/O count based on multiplexing with peripherals
8. All LQFP64 information is indicative and must be confirmed during silicon validation.
9. LBGA208 available only as development package for Nexus2+

2.2.2 Block diagram

Figure 2 shows a top-level block diagram of the SPC560Bx and SPC560Cx family.



Legend:

ADC	Analog-to-Digital Converter
BAM	Boot Assist Module
FlexCAN	Controller Area Network
CMU	Clock Monitor Unit
CTU	Cross Triggering Unit
DSPI	Deserial Serial Peripheral Interface
eMIOS	Enhanced Modular Input Output System
FMPLL	Frequency-Modulated Phase-Locked Loop
I^2C	Inter-integrated Circuit Bus
IMUX	Internal Multiplexer
INTC	Interrupt Controller
JTAG	JTAG controller
LINFlex	Serial Communication Interface (LIN support)
ECSM	Error Correction Status Module
MC_CGM	Clock Generation Module
MC_ME	Mode Entry Module
MC_PCU	Power Control Unit
MC_RGM	Reset Generation Module
MPU	Memory Protection Unit
Nexus	Nexus Development Interface (NDI) Level
NMI	Non-Maskable Interrupt
PIT	Periodic Interrupt Timer
RTC	Real-Time Clock
SIUL	System Integration Unit Lite
SRAM	Static Random-Access Memory
SSCM	System Status Configuration Module
STM	System Timer Module
SWT	Software Watchdog Timer
WKPU	Wakeup Unit

Figure 2. SPC560Bx and SPC560Cx block diagram

2.2.3 Chip-level features

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

- Single issue, 32-bit CPU core complex (e200z0)
  - Compliant with the Power Architecture™ embedded category
  - Includes an instruction set enhancement allowing variable length encoding (VLE) for code size footprint reduction. With the optional encoding of mixed 16-bit and 32-bit instructions, it is possible to achieve significant code size footprint reduction.
- Up to 512 Kbytes on-chip Code Flash supported with the Flash controller
- Up to 64 Kbytes on-chip Data Flash supported with the Flash controller
- Up to 48 Kbytes on-chip SRAM
- Memory protection unit (MPU) with 8 region descriptors and 32-byte region granularity
- Interrupt controller (INTC) capable of handling 148 selectable-priority interrupt sources
- Frequency-modulated phase-locked loop (FMPLL)
- Crossbar switch architecture for concurrent access to peripherals, Flash, or SRAM from multiple bus masters
- Boot assist module (BAM) supports internal Flash programming via a serial link (FlexCAN or LINFlex)
- Timer supports input/output channels providing a range of 16-bit input capture, output compare, and pulse width modulation functions (eMIOS)
- 10-bit analog-to-digital converter (ADC)
- Up to 3 serial peripheral interface (DSPI) modules
- Up to 4 serial communication interface (LINFlex) modules
  - LINFlex 1, 2 and 3: Master capable
  - LINFlex 0: Master capable and slave capable
- Up to 6 enhanced full CAN (FlexCAN) modules with 64 configurable message buffers
- 1 inter-integrated circuit (I^2C) module
- Up to 123 configurable general purpose pins supporting input and output operations (package dependent)
- Real time counter (RTC) with clock source from FIRC or SIRC supporting autonomous wake-up with 1-ms resolution with max timeout of 2 seconds
  - Support for RTC with clock source from SXOSC, supporting wake-up with 1-sec resolution and max timeout of 1 hour
- 6 periodic interrupt timers (PIT) with 32-bit counter resolution
- 1 system module timer (STM)
- Nexus development interface (NDI) per IEEE-ISTO 5001-2003 Class Two Plus
- Device/board boundary scan testing supported with per Joint Test Action Group (JTAG) of IEEE (IEEE 1149.1)
- On-chip voltage regulator (VREG) for regulation of input supply for all internal levels

2.3 Packages

SPC560Bx and SPC560Cx family members are offered in the following package types:

- 64-pin LQFP, 10mm x 10mm outline
- 100-pin LQFP, 0.5mm pitch, 14mm x 14mm outline
- 144-pin LQFP, 0.5mm pitch, 20mm x 20mm outline
- LBGA208, 1mm ball pitch, 17mm x 17mm outline development package

2.4 Developer support

The following development support is available:

- Automotive evaluation boards (EVB) featuring CAN, LIN interfaces, and more
- Compilers
- Debuggers
- JTAG and Nexus interfaces

The following software support is available:

- OSEK solutions will be available from multiple third parties
- CAN and LIN drivers
- AUTOSAR package

3 Memory Map

Table 6 shows the memory map for the SPC560Bx and SPC560Cx. All addresses on the device, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each IP block.

Table 6. SPC560Bx and SPC560Cx memory map

Start address	End address	Size (KB)	Region name
0x0000_0000	0x0000_7FFF	32	Code Flash Sector 0
0x0000_8000	0x0000_BFFF	16	Code Flash Sector 1
0x0000_C000	0x0000_FFFF	16	Code Flash Sector 2
0x0001_0000	0x0001_7FFF	32	Code Flash Sector 3
0x0001_8000	0x0001_FFFF	32	Code Flash Sector 4
0x0002_0000	0x0003_FFFF	128	Code Flash Sector 5
0x0004_0000	0x0005_FFFF	128	Code Flash Sector 6
0x0006_0000	0x0007_FFFF	128	Code Flash Sector 7
0x0008_0000	0x001F_FFFF	1536	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash Shadow Sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash Test Sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_3FFF	16	Data Flash Array 0
0x0080_4000	0x0080_7FFF	16	Data Flash Array 1
0x0080_8000	0x0080_BFFF	16	Data Flash Array 2
0x0080_C000	0x0080_FFFF	16	Data Flash Array 3
0x0081_0000	0x00BF_FFFF	4032	Reserved
0x00C0_0000	0x00C0_3FFF	16	Data test sector
0x00C0_4000	0x00DF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Flash Emulation Mapping
0x2000_0000	0x3FFF_FFFF	524288	Reserved for External Bus Interface
0x4000_0000	0x4000_BFFF	48	SRAM
0x4000_C000	0xC3F8_7FFF	2162160	Reserved
0xC3F8_8000	0xC3F8_BFFF	16	Code Flash A Configuration
0xC3F8_C000	0xC3F8_FFFF	16	Data Flash A Configuration
0xC3F9_0000	0xC3F9_3FFF	16	SIUL
0xC3F9_4000	0xC3F9_7FFF	16	WKPU
0xC3F9_8000	0xC3F9_FFFF	32	Reserved
0xC3FA_0000	0xC3FA_3FFF	16	eMIOS_0

Table 6. SPC560Bx and SPC560Cx memory map (continued)

Start address	End address	Size (KB)	Region name
0xC3FA_4000	0xC3FA_7FFF	16	eMIOS_1
0xC3FA_8000	0xC3FD_7FFF	192	Reserved
0xC3FD_8000	0xC3FD_BFFF	16	SSCM
0xC3FD_C000	0xC3FD_FFFF	16	MC_ME
0xC3FE_0000	0xC3FE_3FFF	16	MC_CGM
0xC3FE_4000	0xC3FE_7FFF	16	MC_RGM
0xC3FE_8000	0xC3FE_BFFF	16	MC_PCU
0xC3FE_C000	0xC3FE_FFFF	16	RTC/API
0xC3FF_0000	0xC3FF_3FFF	16	PIT
0xC3FF4000	0xFFDF_FFFF	981040	Reserved
0xFFE0_0000	0xFFE0_3FFF	16	ADC_0
0xFFE0_4000	0xFFE2_FFFF	176	Reserved
0xFFE3_0000	0xFFE3_3FFF	16	I2C_0
0xFFE3_4000	0xFFE3_FFFF	48	Reserved
0xFFE4_0000	0xFFE4_3FFF	16	LINFlex_0
0xFFE4_4000	0xFFE4_7FFF	16	LINFlex_1
0xFFE4_8000	0xFFE4_BFFF	16	LINFlex_2
0xFFE4_C000	0xFFE4_FFFF	16	LINFlex_3
0xFFE5_0000	0xFFE6_3FFF	80	Reserved
0xFFE6_4000	0xFFE6_7FFF	16	CTU
0xFFE6_8000	0xFFE6_FFFF	32	Reserved
0xFFE7_0000	0xFFE7_3FFF	16	CAN sampler
0xFFE7_4000	0xFFE7_FFFF	48	Reserved
0xFFE8_0000	0xFFEF_FFFF	512	Mirrored range 0x3F80000–0xC3FFFFFF
0xFFF0_0000	0xFFF0_FFFF	64	Reserved
0xFFF1_0000	0xFFF1_3FFF	16	MPU
0xFFF1_4000	0xFFF3_7FFF	144	Reserved
0xFFF3_8000	0xFFF3_BFFF	16	SWT
0xFFF3_C000	0xFFF3_FFFF	16	STM
0xFFF4_0000	0xFFF4_3FFF	16	ECSM
0xFFF4_4000	0xFFF4_7FFF	16	Reserved
0xFFF4_8000	0xFFF4_BFFF	16	INTC
0xFFF4_C000	0xFFF8_FFFF	272	Reserved
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_0
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_1

Table 6. SPC560Bx and SPC560Cx memory map (continued)

Start address	End address	Size (KB)	Region name
0xFFF9_8000	0xFFF9_BFFF	16	DSPI_2
0xFFF9_C000	0xFFFB_FFFF	144	Reserved
0xFFFC_0000	0xFFFC_3FFF	16	FlexCAN_0
0xFFFC_4000	0xFFFC_7FFF	16	FlexCAN_1
0xFFFC_8000	0xFFFC_BFFF	16	FlexCAN_2
0xFFFC_C000	0xFFFC_FFFF	16	FlexCAN_3
0xFFFD_0000	0xFFFD_3FFF	16	FlexCAN_4
0xFFFD_4000	0xFFFD_7FFF	16	FlexCAN_5
0xFFFD_8000	0xFFFF_BFFF	144	Reserved
0xFFFF_C000	0xFFFF_FFFF	16	BAM

4 Signal description

4.1 Introduction

The following sections provide signal descriptions and related information about the functionality and configuration.

4.2 Package pinouts

The LQFP pinouts and the BGA ballmap are provided in the following figures.

For more information on pin multiplexing on this device, see *Table 7* through *Table 10*.

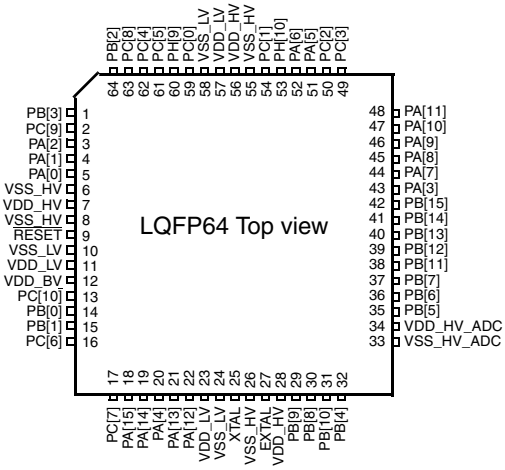
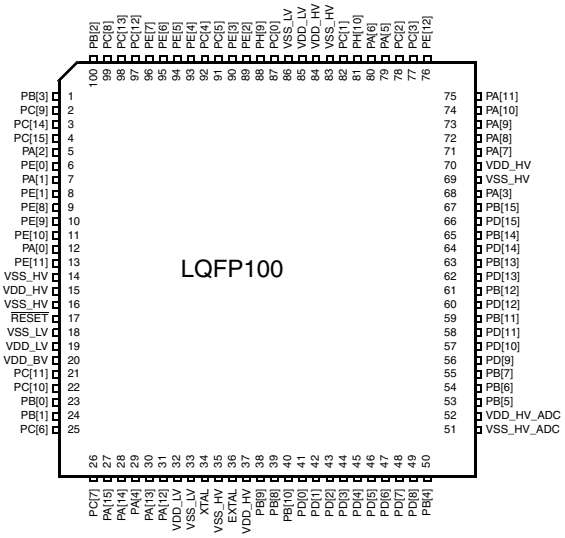


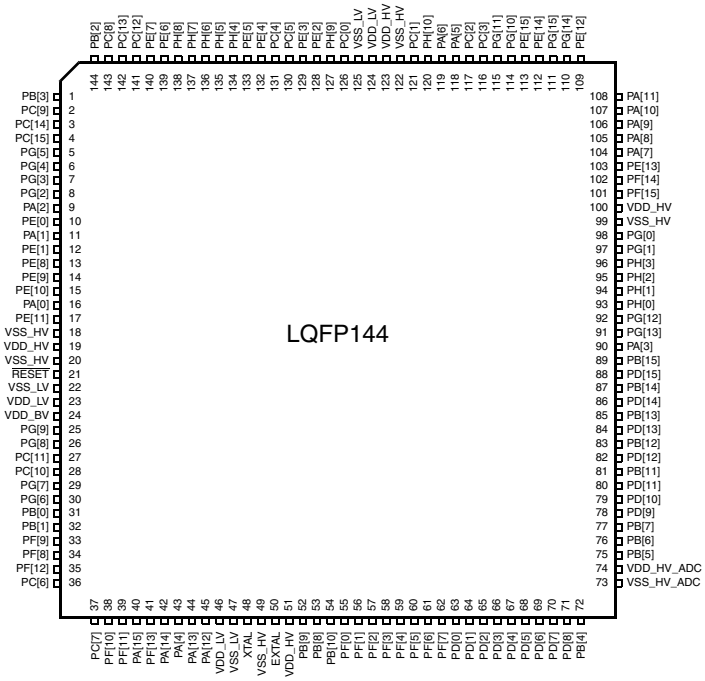
Figure 3. LQFP 64-pin configuration[a]

a. All LQFP64 information is indicative and must be confirmed during silicon validation.



Note:
Availability of port pin alternate functions depends on product selection.

Figure 4. LQFP 100-pin configuration (top view)



Note:
Availability of port pin alternate functions depends on product selection.

Figure 5. LQFP 144-pin configuration (top view)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
A	PC[8]	PC[13]	NC	NC	PH[8]	PH[4]	PC[5]	PC[0]	NC	NC	PC[2]	NC	PE[15]	NC	NC	NC	A
B	PC[9]	PB[2]	NC	PC[12]	PE[6]	PH[5]	PC[4]	PH[9]	PH[10]	NC	PC[3]	PG[11]	PG[15]	PG[14]	PA[11]	PA[10]	B
C	PC[14]	VDD_HV	PB[3]	PE[7]	PH[7]	PE[5]	PE[3]	VSS_LV	PC[1]	NC	PA[5]	NC	PE[14]	PE[12]	PA[9]	PA[8]	C
D	NC	NC	PC[15]	NC	PH[6]	PE[4]	PE[2]	VDD_LV	VDD_HV	NC	PA[6]	NC	PG[10]	PF[14]	PE[13]	PA[7]	D
E	PG[4]	PG[5]	PG[3]	PG[2]									PG[1]	PG[0]	PF[15]	VDD_HV	E
F	PE[0]	PA[2]	PA[1]	PE[1]									PH[0]	PH[1]	PH[3]	PH[2]	F
G	PE[9]	PE[8]	PE[10]	PA[0]			VSS_HV	VSS_HV	VSS_HV	VSS_HV			VDD_HV	NC	NC	MSEO	G
H	VSS_HV	PE[11]	VDD_HV	NC			VSS_HV	VSS_HV	VSS_HV	VSS_HV			MDO3	MDO2	MDO0	MDO1	H
J	RESET	VSS_LV	NC	NC			VSS_HV	VSS_HV	VSS_HV	VSS_HV			NC	NC	NC	NC	J
K	EVTI	NC	VDD_BV	VDD_LV			VSS_HV	VSS_HV	VSS_HV	VSS_HV			NC	PG[12]	PA[3]	PG[13]	K
L	PG[9]	PG[8]	NC	EVTO									PB[15]	PD[15]	PD[14]	PB[14]	L
M	PG[7]	PG[6]	PC[10]	PC[11]									PB[13]	PD[13]	PD[12]	PB[12]	M
N	PB[1]	PF[9]	PB[0]	NC	NC	PA[4]	VSS_LV	EXTAL	VDD_HV	PF[0]	PF[4]	NC	PB[11]	PD[10]	PD[9]	PD[11]	N
P	PF[8]	NC	PC[7]	NC	NC	PA[14]	VDD_LV	XTAL	PB[10]	PF[1]	PF[5]	PD[0]	PD[3]	VDD_HV_ADC	PB[6]	PB[7]	P
R	PF[12]	PC[6]	PF[10]	PF[11]	VDD_HV	PA[15]	PA[13]	NC	OSC32K_XTAL	PF[3]	PF[7]	PD[2]	PD[4]	PD[7]	VSS_HV_ADC	PB[5]	R
T	NC	NC	NC	MCKO	NC	PF[13]	PA[12]	NC	OSC32K_EXTAL	PF[2]	PF[6]	PD[1]	PD[5]	PD[6]	PD[8]	PB[4]	T
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	

Note: LBGA208 available only as development package for Nexus 2+.

NC = Not connected

Figure 6. LBGA208 configuration

4.3 Pad configuration during reset phases

All pads have a fixed configuration under reset.

During the power-up phase, all pads are forced to tristate.

After power-up phase, all pads are forced to tristate with the following exceptions:

- PA[9] (FAB) is pull-down. Without external strong pull-up the device starts fetching from flash.
- PA[8] (ABS[0]) is pull-up.
- RESET pad is driven low. This is pull-up only after PHASE2 reset completion.
- JTAG pads (TCK, TMS and TDI) are pull-up whilst TDO remains tristate.
- Precise ADC pads (PB[7:4] and PD[11:0]) are left tristate (no output buffer available).
- Main oscillator pads (EXTAL, XTAL) are tristate.
- Nexus output pads (MDO[n], MCKO, EVTO, MSEO) are forced to output.

4.4 Voltage supply pins

Voltage supply pins are used to provide power to the device. Two dedicated pins are used for 1.2 V regulator stabilization.

Table 7. Voltage supply pin descriptions

Port pin	Function	Pin number			
		LQFP64	LQFP100	LQFP144	LBGA208(1)
VDD_HV	Digital supply voltage	7, 28, 56	15, 37, 70, 84	19, 51, 100, 123	C2, D9, E16, G13, H3, N9, R5
VSS_HV	Digital ground	6, 8, 26, 55	14, 16, 35, 69, 83	18, 20, 49, 99, 122	G7, G8, G9, G10, H1, H7, H8, H9, H10, J7, J8, J9, J10, K7, K8, K9, K10
VDD_LV	1.2V decoupling pins. Decoupling capacitor must be connected between these pins and the nearest V_{SS_LV} pin.(2)	11, 23, 57	19, 32, 85	23, 46, 124	D8, K4, P7
VSS_LV	1.2V decoupling pins. Decoupling capacitor must be connected between these pins and the nearest V_{DD_LV} pin.(2)	10, 24, 58	18, 33, 86	22, 47, 125	C8, J2, N7
VDD_BV	Internal regulator supply voltage	12	20	24	K3
VSS_HV_ADC	Reference ground and analog ground for the ADC	33	51	73	R15
VDD_HV_ADC	Reference voltage and analog supply for the ADC	34	52	74	P14

1. LBGA208 available only as development package for Nexus2+
2. A decoupling capacitor must be placed between each of the three VDD_LV/VSS_LV supply pairs to ensure stable voltage (see the recommended operating conditions in the device datasheet for details).

4.5 Pad types

In the device the following types of pads are available for system pins and functional port pins:

S = Slow[b]

M = Medium[b] [c]

F = Fast[b] [c]

I = Input only with analog feature[b]

J = Input/Output with analog feature

X = Oscillator

4.6 System pins

The system pins are listed in *Table 8*.

Table 8. System pin descriptions

System pin	Function	I/O direction	Pad type	RESET config.	Pin number			
					LQFP64	LQFP100	LQFP144	LBGA208[1]
$\overline{\text{RESET}}$	Bidirectional reset with Schmitt-Trigger characteristics and noise filter.	I/O	M	Input, weak pull-up only after PHASE2	9	17	21	J1
EXTAL	Analog output 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. [2]	I/O	X	Tristate	27	36	50	N8
XTAL	Analog input of the oscillator amplifier circuit. Needs to be grounded if oscillator is used in bypass mode. [2]	I	X	Tristate	25	34	48	P8

1. LBGA208 available only as development package for Nexus2+
2. See the relevant section of the datasheet

4.7 Functional ports

The functional port pins are listed in *Table 9*.

b. See the I/O pad electrical characteristics in the device datasheet for details.

c. All medium and fast pads are in slow configuration by default at reset and can be configured as fast or medium (see PCR.SRC in *Section Pad Configuration Registers (PCR0–PCR122)*).

Table 9. Functional port pin descriptions

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PA[0]	PCR[0]	AF0 AF1 AF2 AF3 —	GPIO[0] E0UC[0] CLKOUT — WKUP[19][4]	SIUL eMIOS_0 CGL — WKPU	I/O I/O O — I	M	Tristate	5	12	16	G4
PA[1]	PCR[1]	AF0 AF1 AF2 AF3 — —	GPIO[1] E0UC[1] — — NMI[5] WKUP[2][4]	SIUL eMIOS_0 — — WKPU WKPU	I/O I/O — — I I	S	Tristate	4	7	11	F3
PA[2]	PCR[2]	AF0 AF1 AF2 AF3 —	GPIO[2] E0UC[2] — — WKUP[3][4]	SIUL eMIOS_0 — — WKPU	I/O I/O — — I	S	Tristate	3	5	9	F2
PA[3]	PCR[3]	AF0 AF1 AF2 AF3 —	GPIO[3] E0UC[3] — — EIRQ[0]	SIUL eMIOS_0 — — SIUL	I/O I/O — — I	S	Tristate	43	68	90	K15
PA[4]	PCR[4]	AF0 AF1 AF2 AF3 —	GPIO[4] E0UC[4] — — WKUP[9][4]	SIUL eMIOS_0 — — WKPU	I/O I/O — — I	S	Tristate	20	29	43	N6
PA[5]	PCR[5]	AF0 AF1 AF2 AF3	GPIO[5] E0UC[5] — —	SIUL eMIOS_0 — —	I/O I/O — —	M	Tristate	51	79	118	C11
PA[6]	PCR[6]	AF0 AF1 AF2 AF3 —	GPIO[6] E0UC[6] — — EIRQ[1]	SIUL eMIOS_0 — — SIUL	I/O I/O — — I	S	Tristate	52	80	119	D11

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PA[7]	PCR[7]	AF0 AF1 AF2 AF3 —	GPIO[7] E0UC[7] LIN3TX — EIRQ[2]	SIUL eMIOS_0 LINFlex_3 — SIUL	I/O I/O O — I	S	Tristate	44	71	104	D16
PA[8]	PCR[8]	AF0 AF1 AF2 AF3 — N/A[6] —	GPIO[8] E0UC[8] — — EIRQ[3] ABS[0] LIN3RX	SIUL eMIOS_0 — — SIUL BAM LINFlex_3	I/O I/O — — I I I	S	Input, weak pull-up	45	72	105	C16
PA[9]	PCR[9]	AF0 AF1 AF2 AF3 N/A[6]	GPIO[9] E0UC[9] — — FAB	SIUL eMIOS_0 — — BAM	I/O I/O — — I	S	Pull-down	46	73	106	C15
PA[10]	PCR[10]	AF0 AF1 AF2 AF3	GPIO[10] E0UC[10] SDA —	SIUL eMIOS_0 I2C_0 —	I/O I/O I/O —	S	Tristate	47	74	107	B16
PA[11]	PCR[11]	AF0 AF1 AF2 AF3	GPIO[11] E0UC[11] SCL —	SIUL eMIOS_0 I2C_0 —	I/O I/O I/O —	S	Tristate	48	75	108	B15
PA[12]	PCR[12]	AF0 AF1 AF2 AF3 —	GPIO[12] — — — SIN_0	SIUL — — — DSPI0	I/O — — — I	S	Tristate	22	31	45	T7
PA[13]	PCR[13]	AF0 AF1 AF2 AF3	GPIO[13] SOUT_0 — —	SIUL DSPI_0 — —	I/O O — —	M	Tristate	21	30	44	R7
PA[14]	PCR[14]	AF0 AF1 AF2 AF3 —	GPIO[14] SCK_0 CS0_0 — EIRQ[4]	SIUL DSPI_0 DSPI_0 — SIUL	I/O I/O I/O — I	M	Tristate	19	28	42	P6

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PA[15]	PCR[15]	AF0 AF1 AF2 AF3 —	GPIO[15] CS0_0 SCK_0 — WKUP[10][4]	SIUL DSPI_0 DSPI_0 — WKPU	I/O I/O I/O — I	M	Tristate	18	27	40	R6
PB[0]	PCR[16]	AF0 AF1 AF2 AF3	GPIO[16] CAN0TX — —	SIUL FlexCAN_0 — —	I/O O — —	M	Tristate	14	23	31	N3
PB[1]	PCR[17]	AF0 AF1 AF2 AF3 — —	GPIO[17] — — — WKUP[4][4] CAN0RX	SIUL — — — WKPU FlexCAN_0	I/O — — — I I	S	Tristate	15	24	32	N1
PB[2]	PCR[18]	AF0 AF1 AF2 AF3	GPIO[18] LIN0TX SDA —	SIUL LINFlex_0 I2C_0 —	I/O O I/O —	M	Tristate	64	100	144	B2
PB[3]	PCR[19]	AF0 AF1 AF2 AF3 — —	GPIO[19] — SCL — WKUP[11][4] LIN0RX	SIUL — I2C_0 — WKPU LINFlex_0	I/O — I/O — I I	S	Tristate	1	1	1	C3
PB[4]	PCR[20]	AF0 AF1 AF2 AF3 —	GPIO[20] — — — GPI[0]	SIUL — — — ADC	I — — — I	I	Tristate	32	50	72	T16
PB[5]	PCR[21]	AF0 AF1 AF2 AF3 —	GPIO[21] — — — GPI[1]	SIUL — — — ADC	I — — — I	I	Tristate	35	53	75	R16

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PB[6]	PCR[22]	AF0 AF1 AF2 AF3 —	GPIO[22] — — — GPI[2]	SIUL — — — ADC	I — — — I	I	Tristate	36	54	76	P15
PB[7]	PCR[23]	AF0 AF1 AF2 AF3 —	GPIO[23] — — — GPI[3]	SIUL — — — ADC	I — — — I	I	Tristate	37	55	77	P16
PB[8]	PCR[24]	AF0 AF1 AF2 AF3 — —	GPIO[24] — — — ANS[0] OSC32K_XTAL[7]	SIUL — — — ADC SXOSC	I — — — I I/O	I	Tristate	30	39	53	R9
PB[9]	PCR[25]	AF0 AF1 AF2 AF3 — —	GPIO[25] — — — ANS[1] OSC32K_EXTAL[7]	SIUL — — — ADC SXOSC	I — — — I I/O	I	Tristate	29	38	52	T9
PB[10]	PCR[26]	AF0 AF1 AF2 AF3 — —	GPIO[26] — — — ANS[2] WKUP[8][4]	SIUL — — — ADC WKPU	I/O — — — I I	J	Tristate	31	40	54	P9
PB[11][8]	PCR[27]	AF0 AF1 AF2 AF3 —	GPIO[27] E0UC[3] — CS0_0 ANS[3]	SIUL eMIOS_0 — DSPI_0 ADC	I/O I/O — I/O I	J	Tristate	38	59	81	N13
PB[12]	PCR[28]	AF0 AF1 AF2 AF3 —	GPIO[28] E0UC[4] — CS1_0 ANX[0]	SIUL eMIOS_0 — DSPI_0 ADC	I/O I/O — O I	J	Tristate	39	61	83	M16

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PB[13]	PCR[29]	AF0 AF1 AF2 AF3 —	GPIO[29] E0UC[5] — CS2_0 ANX[1]	SIUL eMIOS_0 — DSPI_0 ADC	I/O I/O — O I	J	Tristate	40	63	85	M13
PB[14]	PCR[30]	AF0 AF1 AF2 AF3 —	GPIO[30] E0UC[6] — CS3_0 ANX[2]	SIUL eMIOS_0 — DSPI_0 ADC	I/O I/O — O I	J	Tristate	41	65	87	L16
PB[15]	PCR[31]	AF0 AF1 AF2 AF3 —	GPIO[31] E0UC[7] — CS4_0 ANX[3]	SIUL eMIOS_0 — DSPI_0 ADC	I/O I/O — O I	J	Tristate	42	67	89	L13
PC[0][9]	PCR[32]	AF0 AF1 AF2 AF3	GPIO[32] — TDI —	SIUL — JTAGC —	I/O — I —	M	Input, weak pull-up	59	87	126	A8
PC[1][9]	PCR[33]	AF0 AF1 AF2 AF3	GPIO[33] — TDO[10] —	SIUL — JTAGC —	I/O — O —	M	Tristate	54	82	121	C9
PC[2]	PCR[34]	AF0 AF1 AF2 AF3 —	GPIO[34] SCK_1 CAN4TX[11] — EIRQ[5]	SIUL DSPI_1 LINFlex_4 — SIUL	I/O I/O O — I	M	Tristate	50	78	117	A11
PC[3]	PCR[35]	AF0 AF1 AF2 AF3 — — —	GPIO[35] CS0_1 MA[0] — CAN1RX CAN4RX[11] EIRQ[6]	SIUL DSPI_1 ADC — FlexCAN_1 FlexCAN_4 SIUL	I/O I/O O — I I I	S	Tristate	49	77	116	B11

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PC[4]	PCR[36]	AF0 AF1 AF2 AF3 — —	GPIO[36] — — — SIN_1 CAN3RX[11]	SIUL — — — DSPI_1 FlexCAN_3	I/O — — — I I	M	Tristate	62	92	131	B7
PC[5]	PCR[37]	AF0 AF1 AF2 AF3 —	GPIO[37] SOUT_1 CAN3TX[11] — EIRQ[7]	SIUL DSPI1 FlexCAN_3 — SIUL	I/O O O — I	M	Tristate	61	91	130	A7
PC[6]	PCR[38]	AF0 AF1 AF2 AF3	GPIO[38] LIN1TX — —	SIUL LINFlex_1 — —	I/O O — —	S	Tristate	16	25	36	R2
PC[7]	PCR[39]	AF0 AF1 AF2 AF3 — —	GPIO[39] — — — LIN1RX WKUP[12][4]	SIUL — — — LINFlex_1 WKPU	I/O — — — I I	S	Tristate	17	26	37	P3
PC[8]	PCR[40]	AF0 AF1 AF2 AF3	GPIO[40] LIN2TX — —	SIUL LINFlex_2 — —	I/O O — —	S	Tristate	63	99	143	A1
PC[9]	PCR[41]	AF0 AF1 AF2 AF3 — —	GPIO[41] — — — LIN2RX WKUP[13][4]	SIUL — — — LINFlex_2 WKPU	I/O — — — I I	S	Tristate	2	2	2	B1
PC[10]	PCR[42]	AF0 AF1 AF2 AF3	GPIO[42] CAN1TX CAN4TX[11] MA[1]	SIUL FlexCAN_1 FlexCAN_4 ADC	I/O O O O	M	Tristate	13	22	28	M3

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PC[11]	PCR[43]	AF0 AF1 AF2 AF3 — — —	GPIO[43] — — — CAN1RX CAN4RX[11] WKUP[5][4]	SIUL — — — FlexCAN_1 FlexCAN_4 WKPU	I/O — — — I I I	S	Tristate	—	21	27	M4
PC[12]	PCR[44]	AF0 AF1 AF2 AF3 —	GPIO[44] E0UC[12] — — SIN_2	SIUL eMIOS_0 — — DSPI_2	I/O I/O — — I	M	Tristate	—	97	141	B4
PC[13]	PCR[45]	AF0 AF1 AF2 AF3	GPIO[45] E0UC[13] SOUT_2 —	SIUL eMIOS_0 DSPI_2 —	I/O I/O O —	S	Tristate	—	98	142	A2
PC[14]	PCR[46]	AF0 AF1 AF2 AF3 —	GPIO[46] E0UC[14] SCK_2 — EIRQ[8]	SIUL eMIOS_0 DSPI_2 — SIUL	I/O I/O I/O — I	S	Tristate	—	3	3	C1
PC[15]	PCR[47]	AF0 AF1 AF2 AF3	GPIO[47] E0UC[15] CS0_2 —	SIUL eMIOS_0 DSPI_2 —	I/O I/O I/O —	M	Tristate	—	4	4	D3
PD[0]	PCR[48]	AF0 AF1 AF2 AF3 —	GPIO[48] — — — GPI[4]	SIUL — — — ADC	I — — — I	I	Tristate	—	41	63	P12
PD[1]	PCR[49]	AF0 AF1 AF2 AF3 —	GPIO[49] — — — GPI[5]	SIUL — — — ADC	I — — — I	I	Tristate	—	42	64	T12

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PD[2]	PCR[50]	AF0 AF1 AF2 AF3 —	GPIO[50] — — — GPI[6]	SIUL — — — ADC	I — — — I	I	Tristate	—	43	65	R12
PD[3]	PCR[51]	AF0 AF1 AF2 AF3 —	GPIO[51] — — — GPI[7]	SIUL — — — ADC	I — — — I	I	Tristate	—	44	66	P13
PD[4]	PCR[52]	AF0 AF1 AF2 AF3 —	GPIO[52] — — — GPI[8]	SIUL — — — ADC	I — — — I	I	Tristate	—	45	67	R13
PD[5]	PCR[53]	AF0 AF1 AF2 AF3 —	GPIO[53] — — — GPI[9]	SIUL — — — ADC	I — — — I	I	Tristate	—	46	68	T13
PD[6]	PCR[54]	AF0 AF1 AF2 AF3 —	GPIO[54] — — — GPI[10]	SIUL — — — ADC	I — — — I	I	Tristate	—	47	69	T14
PD[7]	PCR[55]	AF0 AF1 AF2 AF3 —	GPIO[55] — — — GPI[11]	SIUL — — — ADC	I — — — I	I	Tristate	—	48	70	R14
PD[8]	PCR[56]	AF0 AF1 AF2 AF3 —	GPIO[56] — — — GPI[12]	SIUL — — — ADC	I — — — I	I	Tristate	—	49	71	T15

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PD[9]	PCR[57]	AF0 AF1 AF2 AF3 —	GPIO[57] — — — GPI[13]	SIUL — — — ADC	I — — — I	I	Tristate	—	56	78	N15
PD[10]	PCR[58]	AF0 AF1 AF2 AF3 —	GPIO[58] — — — GPI[14]	SIUL — — — ADC	I — — — I	I	Tristate	—	57	79	N14
PD[11]	PCR[59]	AF0 AF1 AF2 AF3 —	GPIO[59] — — — GPI[15]	SIUL — — — ADC	I — — — I	I	Tristate	—	58	80	N16
PD[12][8]	PCR[60]	AF0 AF1 AF2 AF3 —	GPIO[60] CS5_0 E0UC[24] — ANS[4]	SIUL DSPI_0 eMIOS_0 — ADC	I/O O I/O — I	J	Tristate	—	60	82	M15
PD[13]	PCR[61]	AF0 AF1 AF2 AF3 —	GPIO[61] CS0_1 E0UC[25] — ANS[5]	SIUL DSPI_1 eMIOS_0 — ADC	I/O I/O I/O — I	J	Tristate	—	62	84	M14
PD[14]	PCR[62]	AF0 AF1 AF2 AF3 —	GPIO[62] CS1_1 E0UC[26] — ANS[6]	SIUL DSPI_1 eMIOS_0 — ADC	I/O O I/O — I	J	Tristate	—	64	86	L15
PD[15]	PCR[63]	AF0 AF1 AF2 AF3 —	GPIO[63] CS2_1 E0UC[27] — ANS[7]	SIUL DSPI_1 eMIOS_0 — ADC	I/O O I/O — I	J	Tristate	—	66	88	L14

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PE[0]	PCR[64]	AF0 AF1 AF2 AF3 — —	GPIO[64] E0UC[16] — — CAN5RX[11] WKUP[6][4]	SIUL eMIOS_0 — — FlexCAN_5 WKPU	I/O I/O — — I I	S	Tristate	—	6	10	F1
PE[1]	PCR[65]	AF0 AF1 AF2 AF3	GPIO[65] E0UC[17] CAN5TX[11] —	SIUL eMIOS_0 FlexCAN_5 —	I/O I/O O —	M	Tristate	—	8	12	F4
PE[2]	PCR[66]	AF0 AF1 AF2 AF3 —	GPIO[66] E0UC[18] — — SIN_1	SIUL eMIOS_0 — — DSPI_1	I/O I/O — — I	M	Tristate	—	89	128	D7
PE[3]	PCR[67]	AF0 AF1 AF2 AF3	GPIO[67] E0UC[19] SOUT_1 —	SIUL eMIOS_0 DSPI_1 —	I/O I/O O —	M	Tristate	—	90	129	C7
PE[4]	PCR[68]	AF0 AF1 AF2 AF3 —	GPIO[68] E0UC[20] SCK_1 — EIRQ[9]	SIUL eMIOS_0 DSPI_1 — SIUL	I/O I/O I/O — I	M	Tristate	—	93	132	D6
PE[5]	PCR[69]	AF0 AF1 AF2 AF3	GPIO[69] E0UC[21] CS0_1 MA[2]	SIUL eMIOS_0 DSPI_1 ADC	I/O I/O I/O O	M	Tristate	—	94	133	C6
PE[6]	PCR[70]	AF0 AF1 AF2 AF3	GPIO[70] E0UC[22] CS3_0 MA[1]	SIUL eMIOS_0 DSPI_0 ADC	I/O I/O O O	M	Tristate	—	95	139	B5
PE[7]	PCR[71]	AF0 AF1 AF2 AF3	GPIO[71] E0UC[23] CS2_0 MA[0]	SIUL eMIOS_0 DSPI_0 ADC	I/O I/O O O	M	Tristate	—	96	140	C4

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PE[8]	PCR[72]	AF0 AF1 AF2 AF3	GPIO[72] CAN2TX[12] E0UC[22] CAN3TX[11]	SIUL FlexCAN_2 eMIOS_0 FlexCAN_3	I/O O I/O O	M	Tristate	—	9	13	G2
PE[9]	PCR[73]	AF0 AF1 AF2 AF3 — — —	GPIO[73] — E0UC[23] — WKUP[7][4] CAN2RX[12] CAN3RX[11]	SIUL — eMIOS_0 — WKPU FlexCAN_2 FlexCAN_3	I/O — I/O — I I I	S	Tristate	—	10	14	G1
PE[10]	PCR[74]	AF0 AF1 AF2 AF3 —	GPIO[74] LIN3TX CS3_1 — EIRQ[10]	SIUL LINFlex_3 DSPI_1 — SIUL	I/O O O — I	S	Tristate	—	11	15	G3
PE[11]	PCR[75]	AF0 AF1 AF2 AF3 — —	GPIO[75] — CS4_1 — LIN3RX WKUP[14][4]	SIUL — DSPI_1 — LINFlex_3 WKPU	I/O — O — I I	S	Tristate	—	13	17	H2
PE[12]	PCR[76]	AF0 AF1 AF2 AF3 — —	GPIO[76] — E1UC[19][13] — SIN_2 EIRQ[11]	SIUL — eMIOS_1 — DSPI_2 SIUL	I/O — I/O — I I	S	Tristate	—	76	109	C14
PE[13]	PCR[77]	AF0 AF1 AF2 AF3	GPIO[77] SOUT2 E1UC[20] —	SIUL DSPI_2 eMIOS_1 —	I/O O I/O —	S	Tristate	—	—	103	D15
PE[14]	PCR[78]	AF0 AF1 AF2 AF3 —	GPIO[78] SCK_2 E1UC[21] — EIRQ[12]	SIUL DSPI_2 eMIOS_1 — SIUL	I/O I/O I/O — I	S	Tristate	—	—	112	C13

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PE[15]	PCR[79]	AF0 AF1 AF2 AF3	GPIO[79] CS0_2 E1UC[22] —	SIUL DSPI_2 eMIOS_1 —	I/O I/O I/O —	M	Tristate	—	—	113	A13
PF[0]	PCR[80]	AF0 AF1 AF2 AF3 —	GPIO[80] E0UC[10] CS3_1 — ANS[8]	SIUL eMIOS_0 DSPI_1 — ADC	I/O I/O O — I	J	Tristate	—	—	55	N10
PF[1]	PCR[81]	AF0 AF1 AF2 AF3 —	GPIO[81] E0UC[11] CS4_1 — ANS[9]	SIUL eMIOS_0 DSPI_1 — I	I/O I/O O — I	J	Tristate	—	—	56	P10
PF[2]	PCR[82]	AF0 AF1 AF2 AF3 —	GPIO[82] E0UC[12] CS0_2 — ANS[10]	SIUL eMIOS_0 DSPI_2 — ADC	I/O I/O I/O — I	J	Tristate	—	—	57	T10
PF[3]	PCR[83]	AF0 AF1 AF2 AF3 —	GPIO[83] E0UC[13] CS1_2 — ANS[11]	SIUL eMIOS_0 DSPI_2 — ADC	I/O I/O O — I	J	Tristate	—	—	58	R10
PF[4]	PCR[84]	AF0 AF1 AF2 AF3 —	GPIO[84] E0UC[14] CS2_2 — ANS[12]	SIUL eMIOS_0 DSPI_2 — ADC	I/O I/O O — I	J	Tristate	—	—	59	N11
PF[5]	PCR[85]	AF0 AF1 AF2 AF3 —	GPIO[85] E0UC[22] CS3_2 — ANS[13]	SIUL eMIOS_0 DSPI_2 — ADC	I/O I/O O — I	J	Tristate	—	—	60	P11
PF[6]	PCR[86]	AF0 AF1 AF2 AF3 —	GPIO[86] E0UC[23] — — ANS[14]	SIUL eMIOS_0 — — ADC	I/O I/O — — I	J	Tristate	—	—	61	T11

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PF[7]	PCR[87]	AF0 AF1 AF2 AF3 —	GPIO[87] — — — ANS[15]	SIUL — — — ADC	I/O — — — I	J	Tristate	—	—	62	R11
PF[8]	PCR[88]	AF0 AF1 AF2 AF3	GPIO[88] CAN3TX[14] CS4_0 CAN2TX	SIUL FlexCAN_3 DSPI_0 FlexCAN_2	I/O O O O	M	Tristate	—	—	34	P1
PF[9]	PCR[89]	AF0 AF1 AF2 AF3 — —	GPIO[89] — CS5_0 — CAN2RX[15] CAN3RX[14]	SIUL — DSPI_0 — FlexCAN_2 FlexCAN_3	I/O — O — I I	S	Tristate	—	—	33	N2
PF[10]	PCR[90]	AF0 AF1 AF2 AF3	GPIO[90] — — —	SIUL — — —	I/O — — —	M	Tristate	—	—	38	R3
PF[11]	PCR[91]	AF0 AF1 AF2 AF3 —	GPIO[91] — — — WKUP[15][4]	SIUL — — — WKPU	I/O — — — I	S	Tristate	—	—	39	R4
PF[12]	PCR[92]	AF0 AF1 AF2 AF3	GPIO[92] E1UC[25] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	35	R1
PF[13]	PCR[93]	AF0 AF1 AF2 AF3 —	GPIO[93] E1UC[26] — — WKUP[16][4]	SIUL eMIOS_1 — — WKPU	I/O I/O — — I	S	Tristate	—	—	41	T6
PF[14]	PCR[94]	AF0 AF1 AF2 AF3	GPIO[94] CAN4TX[11] E1UC[27] CAN1TX	SIUL FlexCAN_4 eMIOS_1 FlexCAN_4	I/O O I/O O	M	Tristate	—	—	102	D14

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PF[15]	PCR[95]	AF0 AF1 AF2 AF3 — — —	GPIO[95] — — — CAN1RX CAN4RX[11] EIRQ[13]	SIUL — — — FlexCAN_1 FlexCAN_4 SIUL	I/O — — — I I I	S	Tristate	—	—	101	E15
PG[0]	PCR[96]	AF0 AF1 AF2 AF3	GPIO[96] CAN5TX[11] E1UC[23] —	SIUL FlexCAN_5 eMIOS_1 —	I/O O I/O —	M	Tristate	—	—	98	E14
PG[1]	PCR[97]	AF0 AF1 AF2 AF3 — —	GPIO[97] — E1UC[24] — CAN5RX[11] EIRQ[14]	SIUL — eMIOS_1 — FlexCAN_5 SIUL	I/O — I/O — I I	S	Tristate	—	—	97	E13
PG[2]	PCR[98]	AF0 AF1 AF2 AF3	GPIO[98] E1UC[11] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	8	E4
PG[3]	PCR[99]	AF0 AF1 AF2 AF3 —	GPIO[99] E1UC[12] — — WKUP[17][4]	SIUL eMIOS_1 — — WKPU	I/O I/O — — I	S	Tristate	—	—	7	E3
PG[4]	PCR[100]	AF0 AF1 AF2 AF3	GPIO[100] E1UC[13] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	6	E1
PG[5]	PCR[101]	AF0 AF1 AF2 AF3 —	GPIO[101] E1UC[14] — — WKUP[18][4]	SIUL eMIOS_1 — — WKPU	I/O I/O — — I	S	Tristate	—	—	5	E2
PG[6]	PCR[102]	AF0 AF1 AF2 AF3	GPIO[102] E1UC[15] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	30	M2

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PG[7]	PCR[103]	AF0 AF1 AF2 AF3	GPIO[103] E1UC[16] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	29	M1
PG[8]	PCR[104]	AF0 AF1 AF2 AF3 —	GPIO[104] E1UC[17] — CS0_2 EIRQ[15]	SIUL eMIOS_1 — DSPI_2 SIUL	I/O I/O — I/O I	S	Tristate	—	—	26	L2
PG[9]	PCR[105]	AF0 AF1 AF2 AF3	GPIO[105] E1UC[18] — SCK_2	SIUL eMIOS_1 — DSPI_2	I/O I/O — I/O	S	Tristate	—	—	25	L1
PG[10]	PCR[106]	AF0 AF1 AF2 AF3	GPIO[106] E0UC[24] — —	SIUL eMIOS_0 — —	I/O I/O — —	S	Tristate	—	—	114	D13
PG[11]	PCR[107]	AF0 AF1 AF2 AF3	GPIO[107] E0UC[25] — —	SIUL eMIOS_0 — —	I/O I/O — —	M	Tristate	—	—	115	B12
PG[12]	PCR[108]	AF0 AF1 AF2 AF3	GPIO[108] E0UC[26] — —	SIUL eMIOS_0 — —	I/O I/O — —	M	Tristate	—	—	92	K14
PG[13]	PCR[109]	AF0 AF1 AF2 AF3	GPIO[109] E0UC[27] — —	SIUL eMIOS_0 — —	I/O I/O — —	M	Tristate	—	—	91	K16
PG[14]	PCR[110]	AF0 AF1 AF2 AF3	GPIO[110] E1UC[0] — —	SIUL eMIOS_1 — —	I/O I/O — —	S	Tristate	—	—	110	B14
PG[15]	PCR[111]	AF0 AF1 AF2 AF3	GPIO[111] E1UC[1] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	111	B13

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PH[0]	PCR[112]	AF0 AF1 AF2 AF3 —	GPIO[112] E1UC[2] — — SIN1	SIUL eMIOS_1 — — DSPI_1	I/O I/O — — I	M	Tristate	—	—	93	F13
PH[1]	PCR[113]	AF0 AF1 AF2 AF3	GPIO[113] E1UC[3] SOUT1 —	SIUL eMIOS_1 DSPI_1 —	I/O I/O O —	M	Tristate	—	—	94	F14
PH[2]	PCR[114]	AF0 AF1 AF2 AF3	GPIO[114] E1UC[4] SCK_1 —	SIUL eMIOS_1 DSPI_1 —	I/O I/O I/O —	M	Tristate	—	—	95	F16
PH[3]	PCR[115]	AF0 AF1 AF2 AF3	GPIO[115] E1UC[5] CS0_1 —	SIUL eMIOS_1 DSPI_1 —	I/O I/O I/O —	M	Tristate	—	—	96	F15
PH[4]	PCR[116]	AF0 AF1 AF2 AF3	GPIO[116] E1UC[6] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	—	134	A6
PH[5]	PCR[117]	AF0 AF1 AF2 AF3	GPIO[117] E1UC[7] — —	SIUL eMIOS_1 — —	I/O I/O — —	S	Tristate	—	—	135	B6
PH[6]	PCR[118]	AF0 AF1 AF2 AF3	GPIO[118] E1UC[8] — MA[2]	SIUL eMIOS_1 — ADC	I/O I/O — O	M	Tristate	—	—	136	D5
PH[7]	PCR[119]	AF0 AF1 AF2 AF3	GPIO[119] E1UC[9] CS3_2 MA[1]	SIUL eMIOS_1 DSPI_2 ADC	I/O I/O O O	M	Tristate	—	—	137	C5
PH[8]	PCR[120]	AF0 AF1 AF2 AF3	GPIO[120] E1UC[10] CS2_2 MA[0]	SIUL eMIOS_1 DSPI_2 ADC	I/O I/O O O	M	Tristate	—	—	138	A5

Table 9. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET configuration	Pin number			
								LQFP64	LQFP100	LQFP144	LBGA208[3]
PH[9][9]	PCR[121]	AF0 AF1 AF2 AF3	GPIO[121] — TCK —	SIUL — JTAGC —	I/O — I —	S	Input, weak pull-up	—	88	127	B8
PH[10][9]	PCR[122]	AF0 AF1 AF2 AF3	GPIO[122] — TMS —	SIUL — JTAGC —	I/O — I —	S	Input, weak pull-up	—	81	120	B9

1. Alternate functions are chosen by setting the values of the PCR.PA bitfields inside the SIUL module. PCR.PA = 00 → AF0; PCR.PA = 01 → AF1; PCR.PA = 10 → AF2; PCR.PA = 11 → AF3. This is intended to select the output functions; to use one of the input 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 "—".
2. Multiple inputs are routed to all respective modules internally. The input of some modules must be configured by setting the values of the PSMIO.PADSELx bitfields inside the SIUL module.
3. LBGA208 available only as development package for Nexus2+
4. All WKUP pins also support external interrupt capability. See wakeup unit chapter for further details.
5. NMI has higher priority than alternate function. When NMI is selected, the PCR.AF field is ignored.
6. "Not applicable" because these functions are available only while the device is booting. Refer to BAM chapter of the reference manual for details.
7. Value of PCR.IBE bit must be 0
8. Be aware that this pad is used on the SPC560B64L3 and SPC560B64L5 to provide VDD_HV_ADC and VSS_HV_ADC1. Therefore, you should be careful in ensuring compatibility between SPC560Bx and SPC560Cx and SPC560B64.
9. Out of reset all the functional pins except PC[0:1] and PH[9:10] are available to the user as GPIO.
   PC[0:1] are available as JTAG pins (TDI and TDO respectively).
   PH[9:10] are available as JTAG pins (TCK and TMS respectively).
   If the user configures these JTAG pins in GPIO mode the device is no longer compliant with IEEE 1149.1-2001.
10. The TDO pad has been moved into the STANDBY domain in order to allow low-power debug handshaking in STANDBY mode. However, no pull-resistor is active on the TDO pad while in STANDBY mode. At this time the pad is configured as an input. When no debugger is connected the TDO pad is floating causing additional current consumption. To avoid the extra consumption TDO must be connected. An external pull-up resistor in the range of 47–100 kΩ should be added between the TDO pin and VDD. Only in case the TDO pin is used as application pin and a pull-up cannot be used then a pull-down resistor with the same value should be used between TDO pin and GND instead.
11. Available only on SPC560Cx versions and SPC560B50B2 devices
12. Not available on SPC560B40L3 and SPC560B40L5 devices
13. Not available in 100 LQFP package
14. Available only on SPC560B50B2 devices

4.8 Nexus 2+ pins

In the LBGA208 package, eight additional debug pins are available (see *Table 10*).

Table 10. Nexus 2+ pin descriptions

Debug pin	Function	I/O direction	Pad type	Function after reset	Pin number		
					LQFP 100	LQFP 144	LBGA 208(1)
MCKO	Message clock out	O	F	—	—	—	T4
MDO0	Message data out 0	O	M	—	—	—	H15
MDO1	Message data out 1	O	M	—	—	—	H16
MDO2	Message data out 2	O	M	—	—	—	H14
MDO3	Message data out 3	O	M	—	—	—	H13
EVTI	Event in	I	M	Pull-up	—	—	K1
EVTO	Event out	O	M	—	—	—	L4
MSEO	Message start/end out	O	M	—	—	—	G16

1. LBGA208 available only as development package for Nexus2+

5 Microcontroller Boot

This chapter explains the process of booting the microcontroller. The following entities are involved in the boot process:

- *Boot Assist Module (BAM)*
- *System Status and Configuration Module (SSCM)*
- Flash memory boot sectors (see *28, Flash Memory*)
- Memory Management Unit (MMU)

5.1 Boot mechanism

This section describes the configuration required by the user, and the steps performed by the microcontroller, in order to achieve a successful boot from flash memory or serial download modes.

There are 2 external pins on the microcontroller that are latched during reset and used to determine whether the microcontroller will boot from flash memory or attempt a serial download via FlexCAN or LINFlex (RS232):

- FAB (Force Alternate Boot mode) on pin PA[9]
- ABS (Alternate Boot Select) on pin PA[8]

Table 11 describes the configuration options.

Table 11. Boot mode selection

Mode	FAB pin (PA[9])	ABS pin (PA[8])
Flash memory boot (default mode)	0	X
Serial boot (LINFlex)	1	0
Serial boot (FlexCAN)	1	1

The microcontroller has a weak pull-down on PA[9] and a weak pull-up on PA[8]. This means that if nothing external is connected to these pins, the microcontroller will enter flash memory boot mode by default. In order to change the boot behavior, you should use external pullup or pulldown resistors on PA[9] and PA[8]. If there is any external circuitry connected to either pin, you must ensure that this does not interfere with the expected value applied to the pin at reset. Otherwise, the microcontroller may boot into an unexpected mode after reset.

The SSCM preforms a lot of the automated boot activity including reading the latched value of the FAB (PA[9]) pin to determine whether to boot from flash memory or serial boot mode. This is illustrated in *Figure 7*.

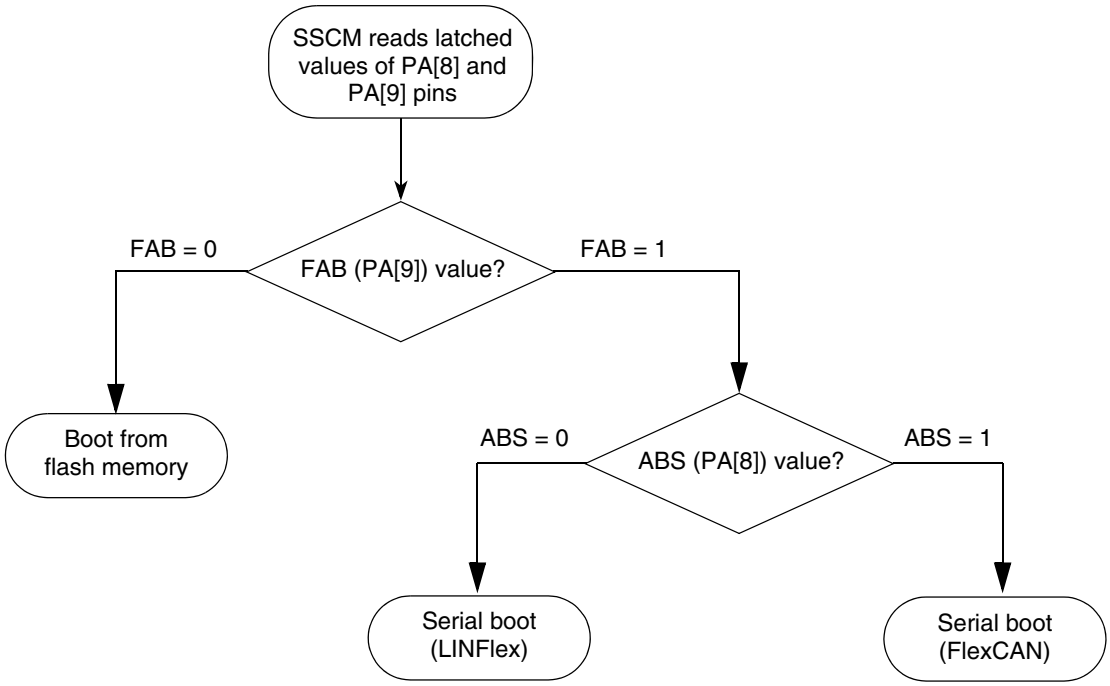


Figure 7. Boot mode selection

5.1.1 Flash memory boot

In order to sucessfully boot from flash memory, you must program two 32-bit fields into one of 5 possible boot blocks as detailed below. The entities to program are:

- 16-bit Reset Configuration Half Word (RCHW), which contains:
  - A BOOT_ID field that must be correctly set to 0x5A in order to "validate" the boot sector
- 32-bit reset vector (this is the start address of the user code)

The location and structure of the boot sectors in flash memory are shown in *Figure 8*.

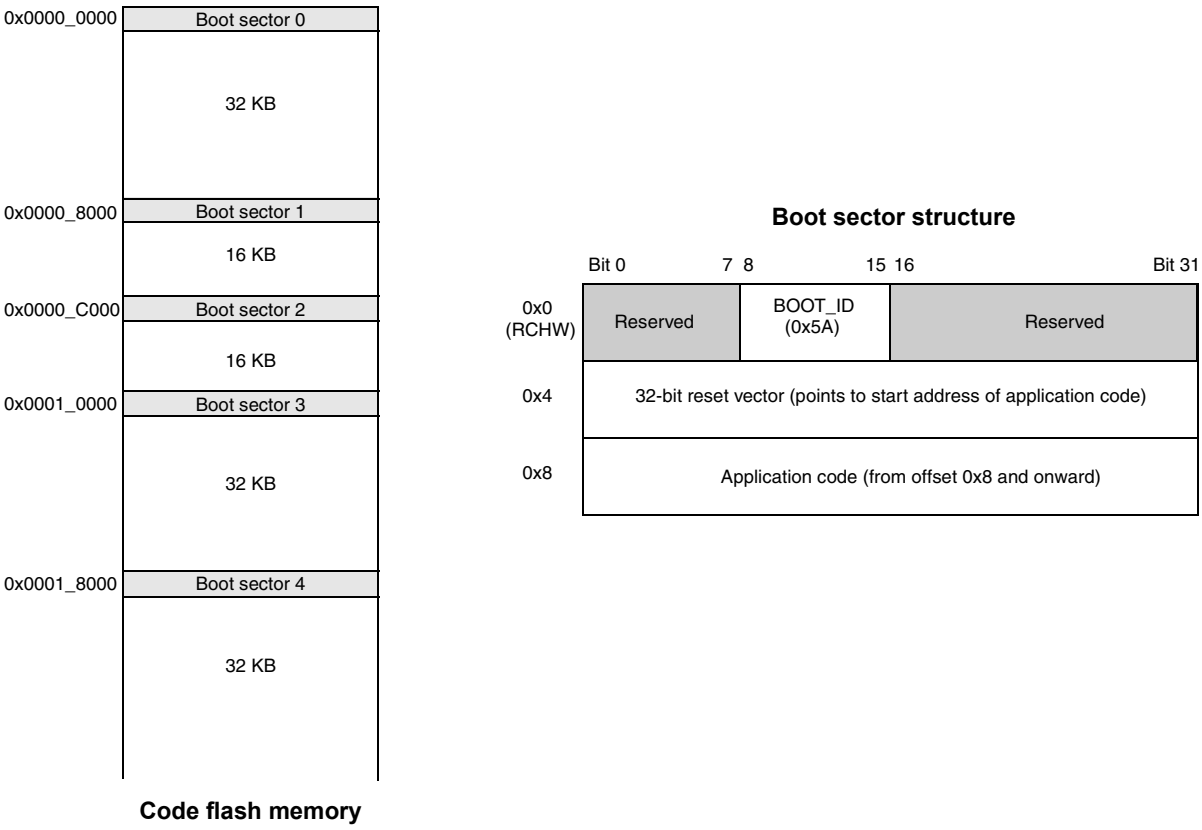


Figure 8. Boot sector structure

The RCHW fields are described in *Table 12*.

Table 12. RCHW field descriptions

Field	Description
BOOT_ID	Boot identifier. If BOOT_ID = 0x5A, the boot sector is considered valid and bootable.

The SSCM performs a sequential search of each boot sector (starting at sector 0) for a valid BOOT_ID within the RCHW. If a valid BOOT_ID is found, the SSCM reads the boot vector address. If a valid BOOT_ID is not found, the SSCM starts the process of putting the microcontroller into static mode.

Finally, the SSCM sets the e200z0h core instruction pointer to the reset vector address and starts the core running.

Static mode

If no valid BOOT_ID within the RCHW was found, the SSCM sets the CPU core instruction pointer to the BAM address and the core starts to execute the code to enter static mode as follows:

- The core executes the "wait" instruction which halts the core.

The sequence is illustrated in *Figure 9*.

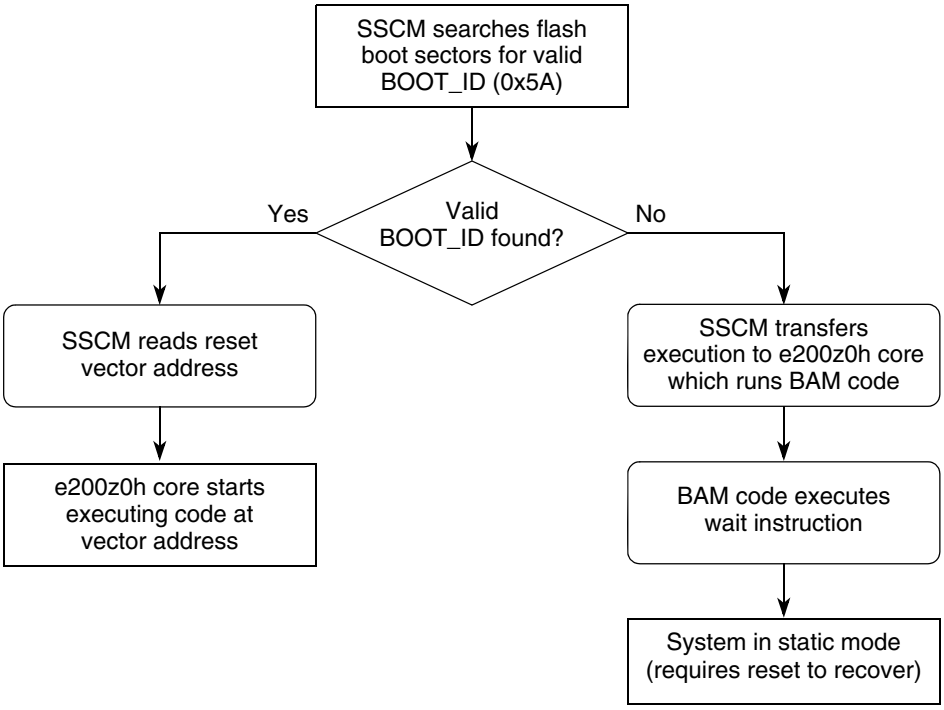


Figure 9. Flash memory boot mode sequence

Alternate boot sectors

Some applications require an alternate boot sector so that the main boot code can be erased and reprogrammed in the field. When an alternate boot is needed, you can create two bootable sectors:

- The valid boot sector located at the lowest address is the main boot sector.
- The valid boot sector located at the next available address is the alternate boot sector.

This scheme ensures that there is always one active boot sector even if the main boot sector is erased.

5.1.2 Serial boot mode

Serial boot provides a mechanism to download and then execute code into the microcontroller SRAM. Code may be downloaded using either FlexCAN or LINFlex (RS232). After the SSCM has detected that serial boot mode has been requested, execution is transferred to the BAM which handles all of the serial boot mode tasks. See *Section 5.2, Boot Assist Module (BAM)*, for more details.

5.1.3 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 321* for a full description):

- Nonvolatile Private Censorship Password registers, NVPWD0 and NVPWD1
- Nonvolatile System Censorship Control registers, NVSCC0 and NVSCC1

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 13*:

Table 13. 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 (NVSCC0 and NVSCC1)

These registers are used together to define the censorship configuration. After factory test these registers are programmed as shown below which disables censorship:

- NVSCC0 = 0x55AA_55AA
- NVSCC1 = 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 NVSCC0,1 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 NVSCC0,1 values.
3. Re-program the shadow flash memory and NVPWD0,1 and NVSCC0,1 registers with your new values. A POR is required before these will take effect.

Caution:

If
(NVSCC0 and NVSCC1 do not match)
or
(Either NVSCC0 or NVSCC1 is not set to 0x55AA)
then the microcontroller will be permanently censored with no way to get back in.

Table 14 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 NVSCC0,1 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 14. Censorship configuration and truth table

Boot configuration		Serial censorship control word (NVSCC*n*[SC])	Censorship control word (NVSCC*n*[CW])	Internal flash memory state	Nexus state	Serial password	JTAG password
FAB pin state	Control options						
0 (flash memory boot)	Uncensored	0xXXXX AND NVSCC0 == NVSCC1	0x55AA AND NVSCC0 == NVSCC1	Enabled	Enabled		N/A
	Private flash memory password and censored	0x55AA AND NVSCC0 == NVSCC1	!0x55AA AND NVSCC0 == NVSCC1	Enabled	Enabled with password		NVPWD**1,0** (SSCM reads flash memory[(1)])
	Censored with no password access (lockout)	!0x55AA	!0X55AA	Enabled	Disabled		N/A
		OR NVSCC0 != NVSCC1					

Table 14. Censorship configuration and truth table (continued)

Boot configuration		Serial censorship control word (NVSCC*n*[SC])	Censorship control word (NVSCC*n*[CW])	Internal flash memory state	Nexus state	Serial password	JTAG password
FAB pin state	Control options						
1 (serial boot)	Private flash memory password and uncensored	0x55AA AND NVSCC0 == NVSCC1		Enabled	Enabled	NVPWD**0,1** (BAM reads flash memory[1])	
	Private flash memory password and censored	0x55AA AND NVSCC0 == NVSCC1	!0x55AA AND NVSCC0 == NVSCC1	Enabled	Disabled	NVPWD**1,0** (SSCM reads flash memory[1])	
	Public password and uncensored	!0x55AA AND NVSCC0 != NVSCC1	0X55AA AND NVSCC0 != NVSCC1	Enabled	Enabled	Public (0xFEED_FACE_CAFE_BEEF)	
	Public password and censored (lockout)	!0x55AA		Disabled	Disabled	Public (0xFEED_FACE_CAFE_BEEF)	
		OR NVSCC0 != NVSCC1					

= 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 10* and *Figure 11* provide a way to quickly check what will happen with different configurations of the NVSCC0,1 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.

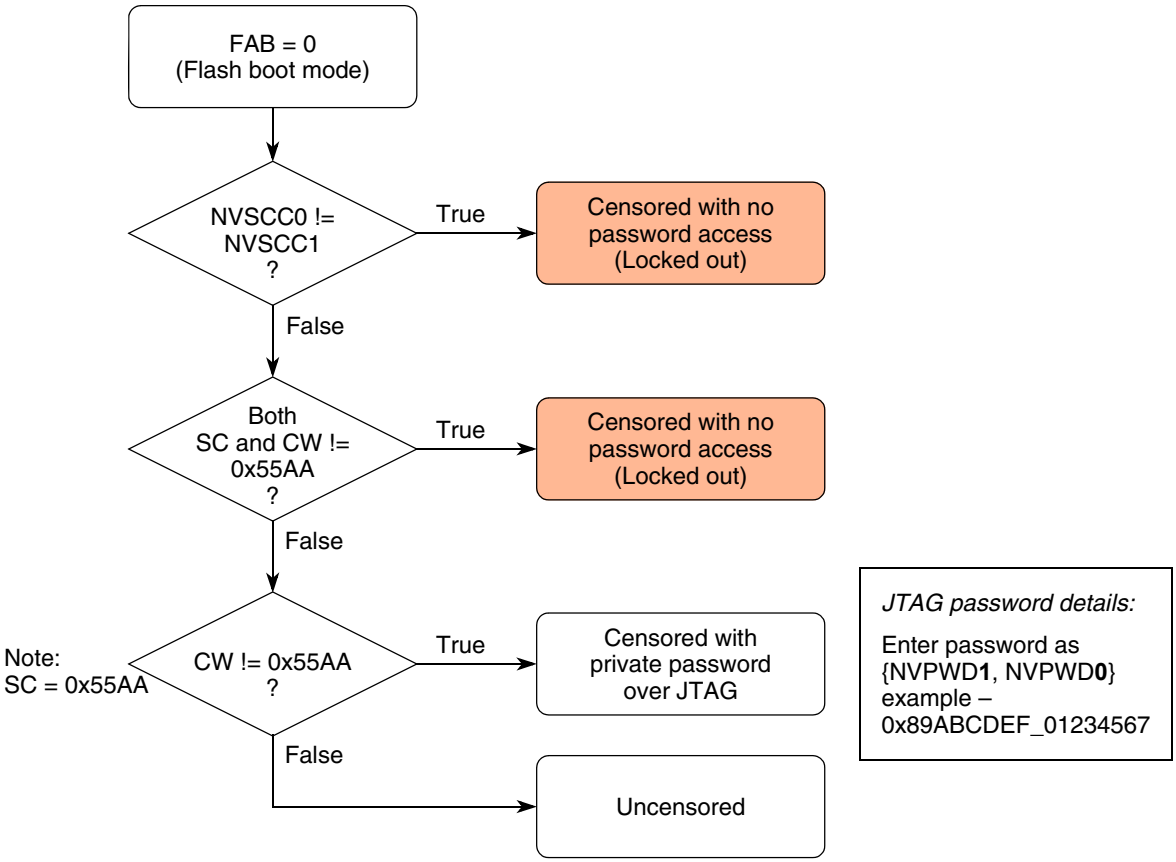


Figure 10. Censorship control in flash memory boot mode

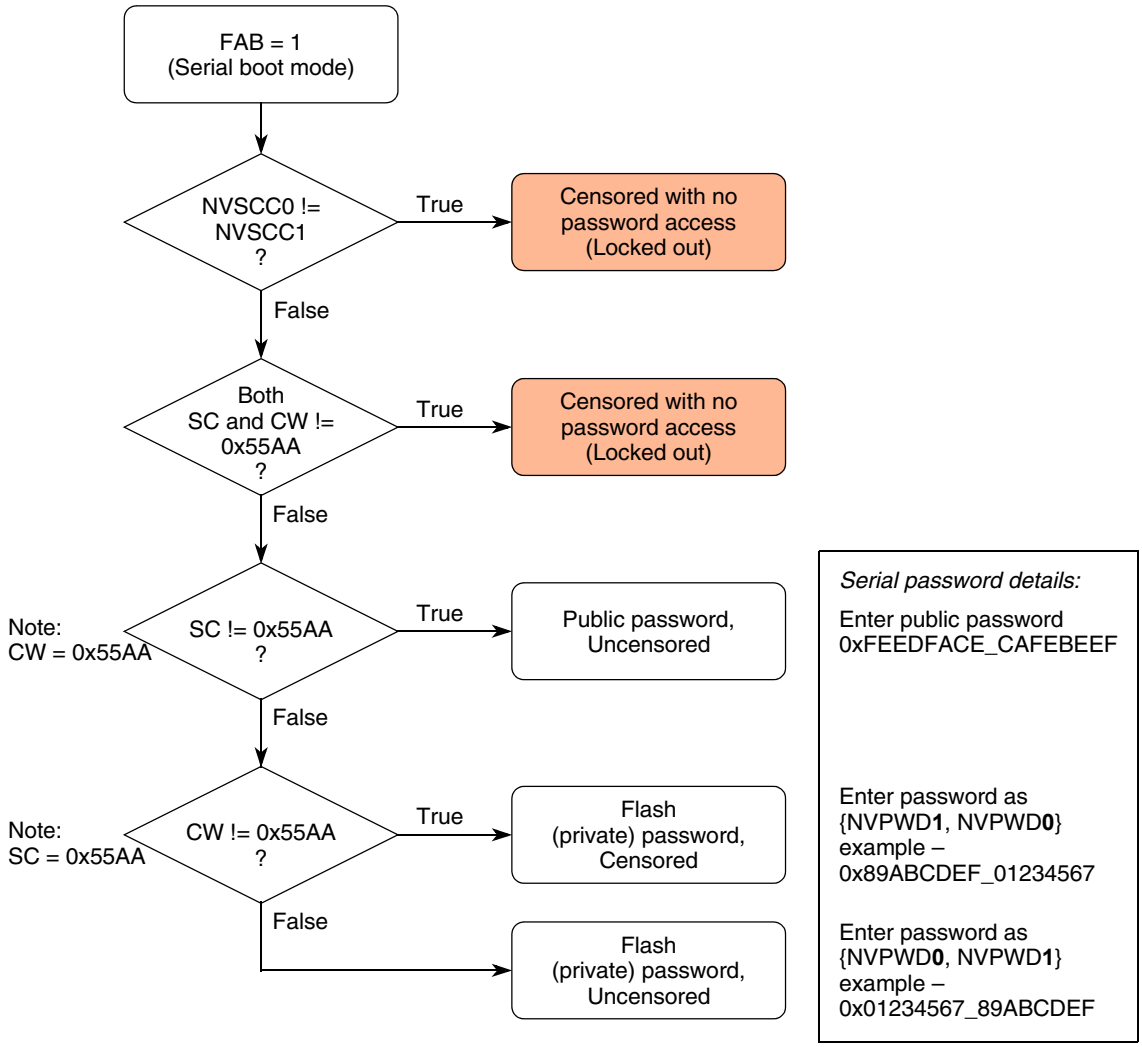


Figure 11. Censorship control in serial boot mode

5.2 Boot Assist Module (BAM)

The BAM consits of a block of ROM at address 0xFFFF_C000 containing VLE firmware. The BAM provides 2 main functions:

- Manages the serial download (FlexCAN or LINFlex protocols supported) including support for a serial password if censorship is enabled
- Places the microcontroller into static mode if flash memory boot mode is selected and a valid BOOT_ID is not located in one of the boot sectors by the SSCM

5.2.1 BAM software flow

Figure 12 illustrates the BAM logic flow.

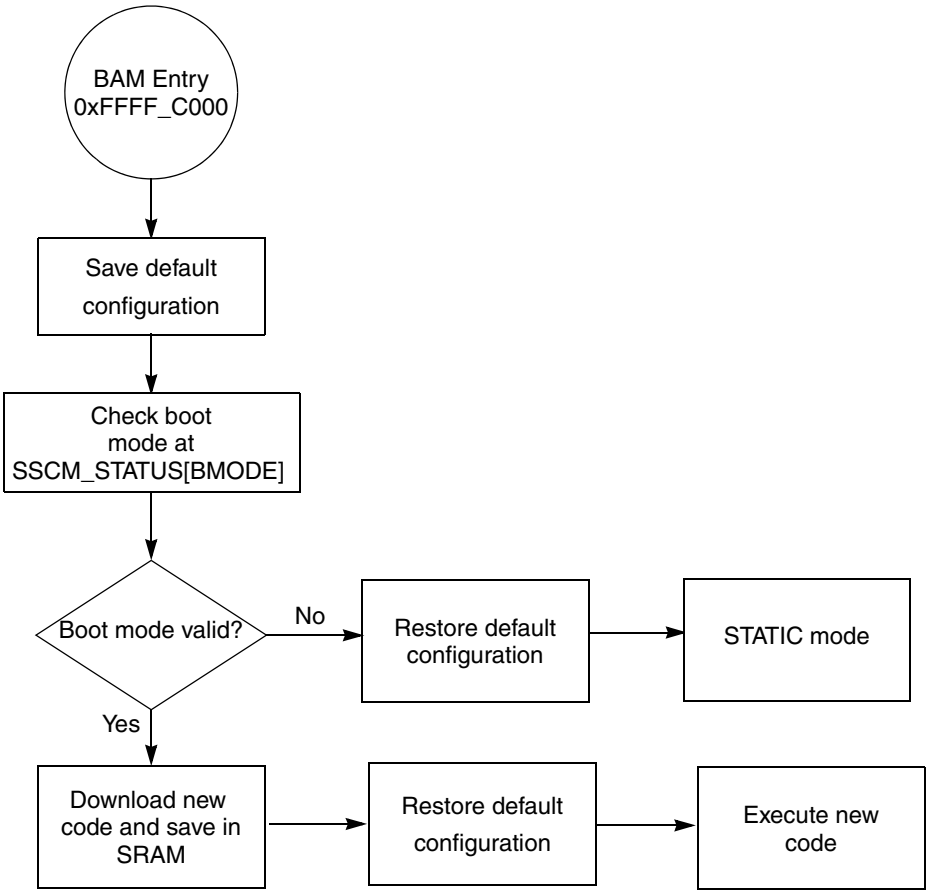


Figure 12. BAM logic flow

The initial (reset) device configuration is saved including the mode and clock configuration. This means that the serial download software running in the BAM can make changes to the modes and clocking and then restore these to the default values before running the newly downloaded application code from the SRAM.

The SSCM_STATUS[BMODE] field indicates which boot mode is to be executed (see *Table 15*). This field is only updated during reset.

There are 2 conditions where the boot mode is not considered valid and the BAM pushes the microcontroller into static mode after restoring the default configuration:

- BMODE = 011 (flash memory boot mode). This means that the SSCM has been unable to find a valid BOOT_ID in the boot sectors so has called the BAM
- BMODE = reserved

In static mode a wait instruction is executed to halt the core.

For the FlexCAN and LINFlex serial boot modes, the respective area of BAM code is executed to download the code to SRAM.

Table 15. SSCM_STATUS[BMODE] values as used by BAM

BMODE value	Corresponding boot mode
000	Reserved
001	FlexCAN_0 serial boot loader
010	LINFlex_0 (RS232 /UART) serial boot loader
011	Flash memory boot mode
100–111	Reserved

After the code has been downloaded to SRAM, the BAM code restores the initial device configuration and then transfers execution to the start address of the downloaded code.

BAM resources

The BAM uses/initializes the following MCU resources:

- MC_ME and MC_CGM to initialize mode and clock sources
- FlexCAN_0, LINFlex _0 and the respective I/O pins when performing serial boot mode
- SSCM and shadow flash memory (NVPWD0,1 and NVSCC0,1) during password check
- SSCM to check the boot mode (see *Table 15*)
- 4–16 MHz fast external crystal oscillator

The system clock is selected directly from the 4–16 MHz fast external crystal oscillator. Thus, the external oscillator frequency defines the baud rates used for serial download (see *Table 16*).

Table 16. Serial boot mode – baud rates

FXOSC frequency (MHz)	LINFlex baud rate (baud)	CAN bit rate (bit/s)
f_{FXOSC}	$f_{FXOSC}/833$	$f_{FXOSC}/40$
8	9600	200K
12	14400	300K
16	19200	400K

Download and execute the new code

From a high level perspective, the download protocol follows these steps:

1. Send the 64-bit password.
2. Send the start address, size of code to be downloaded (in bytes) and the VLE bit[(d)].
3. Download the code.

Each step must be completed before the next step starts. After the download is complete (the specified number of bytes is downloaded), the code executes from the start address.

d. Since the device supports only VLE code and not Book E code, this flag is used only for backward compatibility.

The communication is done in half duplex manner, whereby the transmission from the host is followed by the microcontroller transmission mirroring the transmission back to the host:

- Host sends data to the microcontroller and waits for a response.
- MCU echoes to host the data received.
- Host verifies if echo is correct:
  - If data is correct, the host can continue to send data.
  - If data is not correct, the host stops transmission and the microcontroller enters static mode.

All multi-byte data structures are sent with MSB first.

A more detailed description of these steps follows.

Censorship mode detection and serial password validation

Before the serial download can commence, the BAM code must determine which censorship mode the microcontroller is in and which password to use. It does this by reading the PUB and SEC fields in the SSCM Status Register (see *Section , System Status Register (SSCM_STATUS)*) as shown in *Table 17*.

Table 17. BAM censorship mode detection

SSCM_STATUS register fields		Mode	Password comparison
PUB	SEC		
1	0	Uncensored, public password	0xFEED_FACE_CAFE_BEEF
0	0	Uncensored, private password	NVPWD**0,1** from flash memory via BAM
0	1	Censored, private password	NVPWD**1,0** from flash memory via SSCM

When censorship is enabled, the flash memory cannot be read by application code running in the BAM or in the SRAM. This means that the private password in the shadow flash memory cannot be read by the BAM code. In this case the SSCM is used to obtain the private password from the flash memory of the censored device. When the SSCM reads the private password it inverts the order of {NVPWD**0**, NWPWD**1**} so the password entered over the serial download needs to be {NVPWD**1**, NVPWD**0**}.

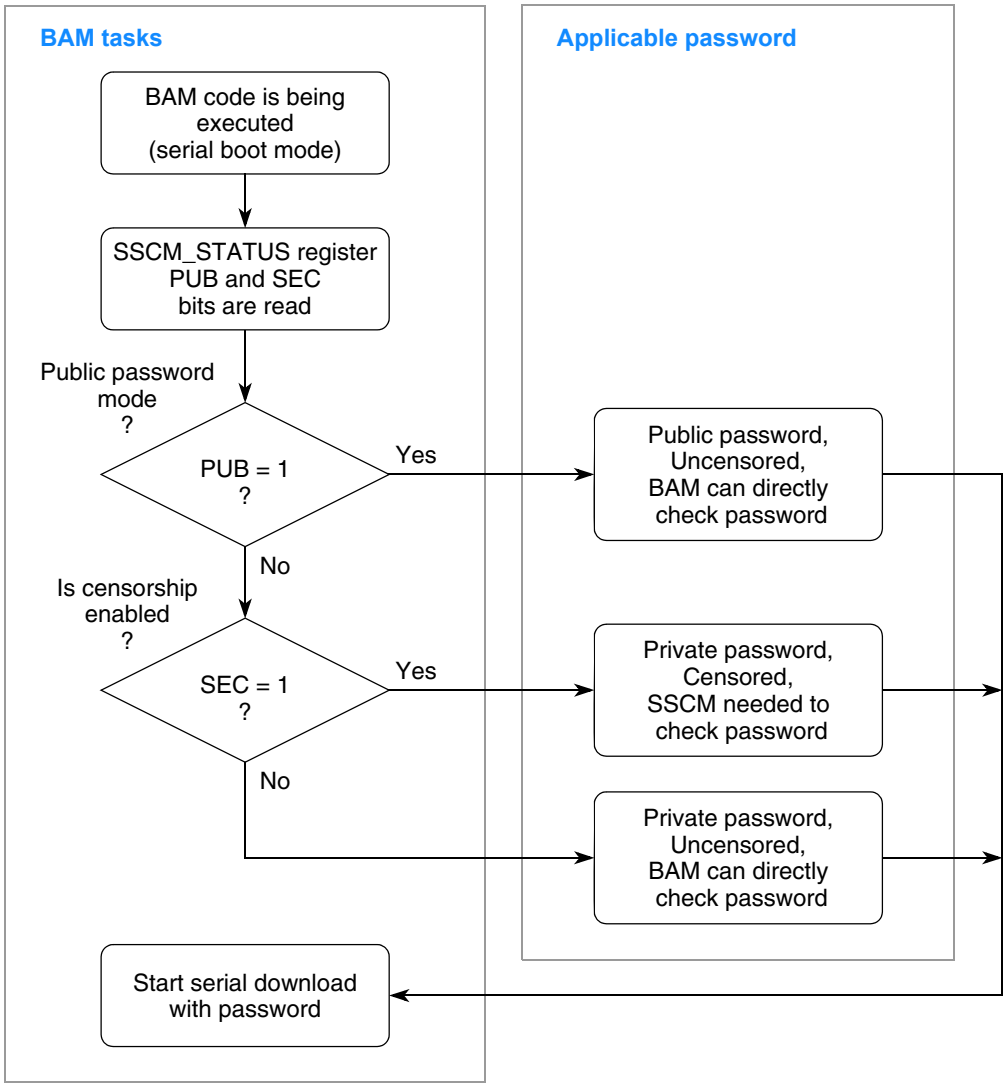


Figure 13. BAM censorship mode detection

The first thing to be downloaded is the 64-bit password. If the password does not match the stored password, then the BAM code pushes the microcontroller into static mode.

The way the password is compared with either the public or private password (depending on mode) varies depending on whether censorship is enabled as described in the following subsections.

Censorship disabled (private or public passwords):

1. If the public password is used, the BAM code does a direct comparison between the serial password and 0xFEED_FACE_CAFE_BEEF.
2. If the private password is used, the BAM code does a direct comparison between the serial password and the private password in flash memory, {NVPWD**0**, NVPWD**1**}.
3. If the password does not match, the BAM code immediately terminates the download and pushes the microcontroller into static mode.

Censorship enabled (private password)

1. Since the flash is secured, the SSCM is required to read the private password.
2. The BAM code writes the serial password to the SSCM_PWCMPH and SSCM_PWCMPL registers.
3. The BAM code then continues with the serial download (start address, data size and data) until all the data has been copied to the SRAM.
4. In the meantime the SSCM has compared the private password in flash with the serial download password the BAM code wrote into SSCM_PWCMPH and SSCM_PWCMPL.
5. If the SSCM obtains a match in the passwords, the censorship is temporarily disabled (until the next reset).
6. The SSCM updates the status of the security (SEC) bit to reflect whether the passwords matched (SEC = 0) or not (SEC = 1)
7. Finally, the BAM code reads SEC. If SEC = 0, execution is transferred to the code in the SRAM. If SEC = 1, the BAM code forces the microcontroller into static mode.

Figure 14 shows this in more detail.

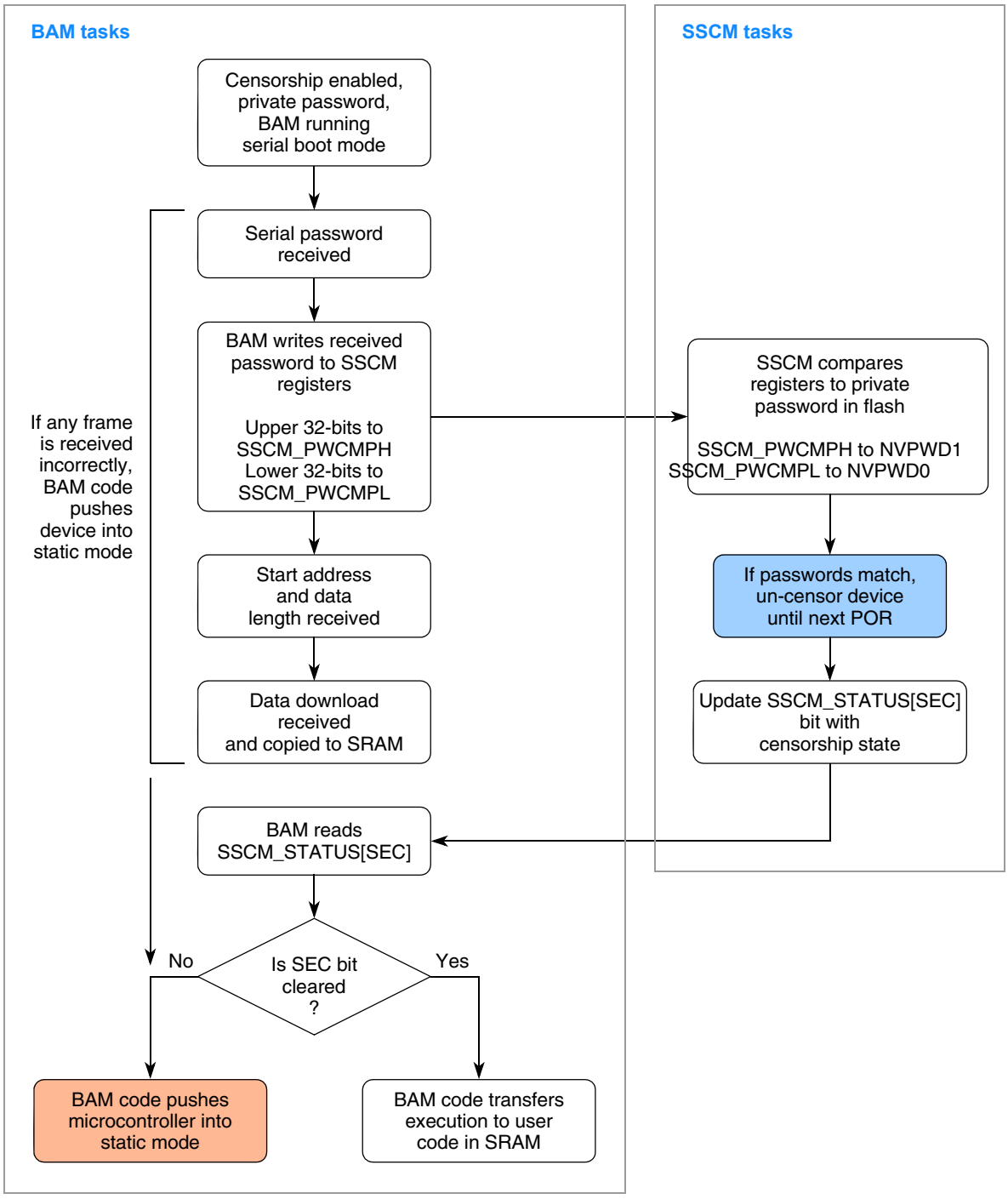


Figure 14. BAM serial boot mode flow for censorship enabled and private password

With LINFlex, any receive error will result in static mode. With FlexCAN, the host will re-transmit data if there has been no acknowledgment from the microcontroller. However there could be a situation where the receiver configuration has an error which would result in static mode entry.

Note: In a censored device booting with serial boot mode, it is possible to read the content of the four 32-bit flash memory locations that make up the boot sector. For example, if the RCHW is stored at address 0x0000_0000, the reads at address 0x0000_0000, 0x0000_0004, 0x0000_0008 and 0x0000_000C will return a correct value. No other flash memory locations can be read.

Download start address, VLE bit and code size

The next 8 bytes received by the microcontroller contain a 32-bit Start Address, the VLE mode bit and a 31-bit code Length as shown in *Figure 15*.

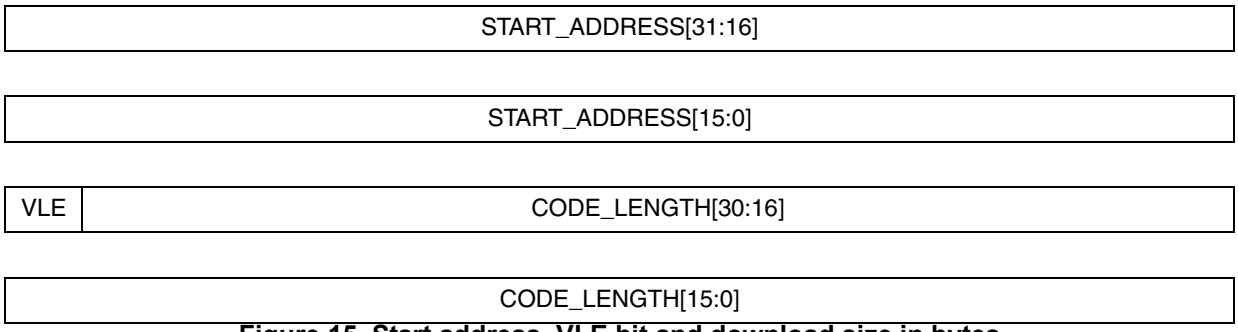


Figure 15. Start address, VLE bit and download size in bytes

The VLE bit (Variable Length Instruction) is used to indicate whether the code to be downloaded is Book VLE or Book III-E. 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 start address is 32-bit word aligned and the 2 least significant bits are ignored by the BAM code.

Note: The start address is configurable, but most not lie within the 0x4000_0000 to 0x4000_00FF address range.

The Length defines how many data bytes have to be loaded.

Download data

Each byte of data received is stored in the microcontroller'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 by the code length.

Since the SRAM is protected by 32-bit wide Error Correction Code (ECC), the BAM code always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, the BAM code fills any additional bytes with 0x0.

Since the ECC on the SRAM has not been initialized (except for the bytes of data that have just been downloaded), an additional dummy word of 0x0000_0000 is written at the end of the downloaded data block to avoid any ECC errors during core prefetch.

Execute code

The BAM code waits for the last data byte to be received. If the operating mode is censored with a private password, then the BAM reads the SSCM status register to determine

whether the serial password matched the private password. If there was a password match then the BAM code restores the initial configuration and transfers execution to the downloaded code start address in SRAM. If the passwords did not match, the BAM code forces a static mode entry.

Note: The watchdog is disabled at the start of BAM code execution. In the case of an unexpected issue during BAM code execution, the microcontroller may be stalled and an external reset required to recover the microcontroller.

5.2.2 LINFlex (RS232) boot

Configuration

Boot according to the LINFlex boot mode download protocol (see *Section , Protocol*) is performed by the LINFlex_0 module in UART (RS232) mode. Pins used are:

- LIN0TX mapped on PB[2]
- LIN0RX mapped on PB[3]

Boot from LINFlex uses the system clock driven by the 4–16 MHz external crystal oscillator (FXOSC).

The LINFlex controller is configured to operate at a baud rate = system clock frequency/833, using an 8-bit data frame without parity bit and 1 stop bit.

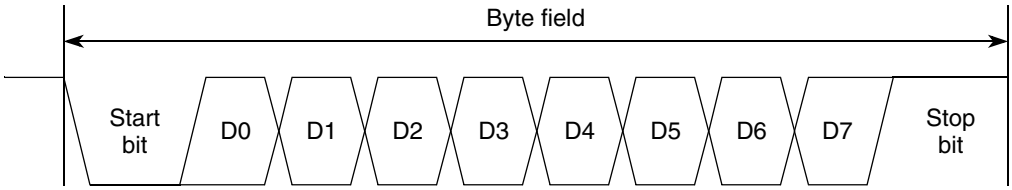


Figure 16. LINFlex bit timing in UART mode

Protocol

Table 18 summarizes the protocol and BAM action during this boot mode.

Table 18. UART boot mode download protocol

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 are stored for future use. Verify if VLE bit is set to 1

Table 18. UART boot mode download protocol

Protocol step	Host sent message	BAM response message	Action
4	8 bits of raw binary data	8 bits of raw binary data	8-bit data are packed into a 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

5.2.3 FlexCAN boot

Configuration

Boot according to the FlexCAN boot mode download protocol (see *Section , Protocol*) is performed by the FlexCAN_0 module. Pins used are:

- CAN0TX mapped on PB[0]
- CAN0RX mapped on PB[1]

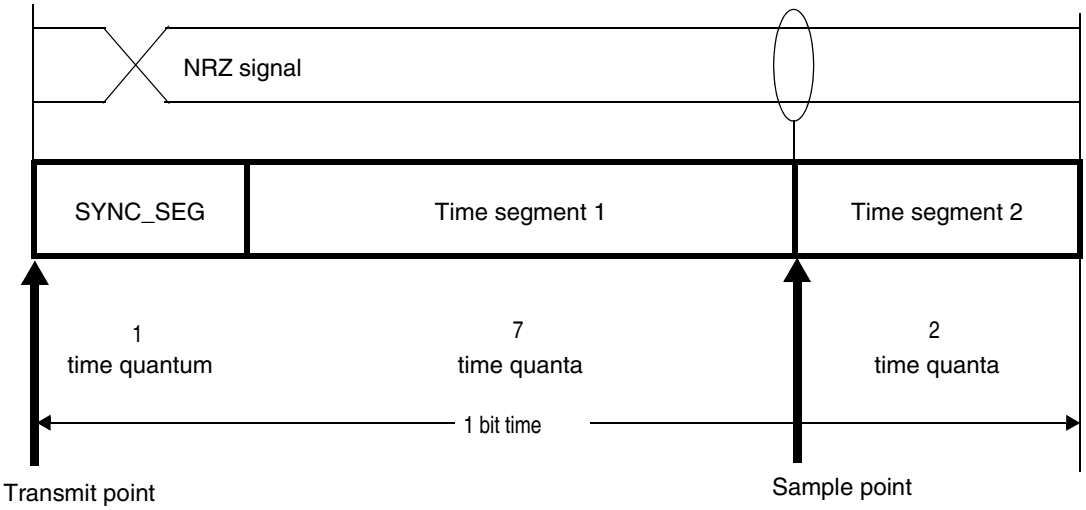
Note: When the serial download via FlexCAN is selected and the device is part of a CAN network, the serial download may stop unexpectedly if there is any other traffic on the network. To avoid this situation, ensure that no other CAN device on the network is active during the serial download process.

Boot from FlexCAN uses the system clock driven by the 4–16 MHz fast external crystal oscillator.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency/40 (see *Table 16* for examples of baud rate).

It uses the standard 11-bit identifier format detailed in 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 17*.



1 time quantum = 4 system clock periods

Figure 17. FlexCAN bit timing

Protocol

Table 19 summarizes the protocol and BAM action during this boot mode. All data are transmitted byte wise.

Table 19. FlexCAN boot mode download protocol

Protocol step	Host sent message	BAM response message	Action
1	CAN ID 0x011 + 64-bit password	CAN ID 0x001 + 64-bit password	Password checked for validity and compared against stored password
2	CAN ID 0x012 + 32-bit store address + VLE bit + 31-bit number of bytes	CAN ID 0x002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address is stored for future use. Size of download are stored for future use. Verify if VLE bit is set to 1
3	CAN ID 0x013 + 8 to 64 bits of raw binary data	CAN 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.
5	None	None	Branch to downloaded code

5.3 System Status and Configuration Module (SSCM)

5.3.1 Introduction

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.

On microcontrollers with a separate STANDBY power domain, the System Status block is part of that domain.

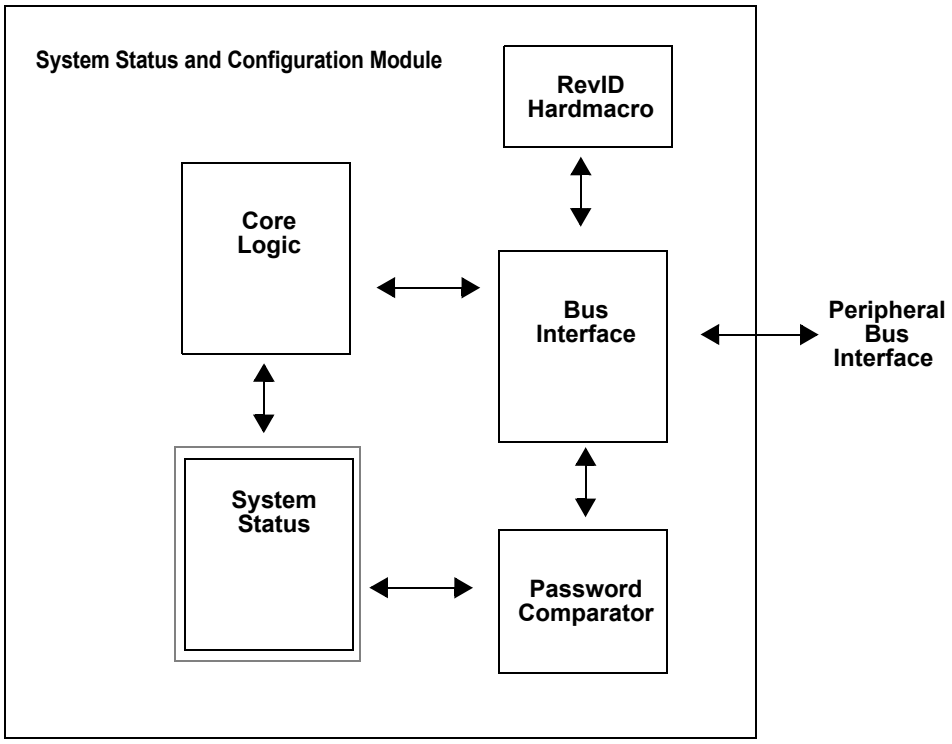


Figure 18. SSCM block diagram

5.3.2 Features

The SSCM includes these features:

- System Configuration and Status
  - Memory sizes/status
  - Microcontroller Mode and Security Status (including censorship and serial boot information)
  - Search Code Flash for bootable sector
  - Determine boot vector
- Device identification information (MCU ID Registers)
- Debug Status Port enable and selection
- Bus and peripheral abort enable/disable

5.3.3 Modes of operation

The SSCM operates identically in all system modes.

5.3.4 Memory map and register description

Table 20 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 20. SSCM memory map

Address offset	Register	Location
0x00	System Status Register (SSCM_STATUS)	*on page 5-99*
0x02	System Memory Configuration Register (SSCM_MEMCONFIG)	*on page 5-100*
0x04	Reserved	
0x06	Error Configuration (SSCM_ERROR)	*on page 5-101*
0x08	Debug Status Port Register (SSCM_DEBUGPORT)	*on page 5-102*
0x0A	Reserved	
0x0C	Password Comparison Register High Word (SSCM_PWCMPH)	*on page 5-104*
0x10	Password Comparison Register Low Word (SSCM_PWCMPL)	*on page 5-104*

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, the SSCM_STATUS register is accessible by a 16-bit read/write to address ‘Base + 0x0002’, but performing a 16-bit access to ‘Base + 0x0003’ is illegal.

System Status Register (SSCM_STATUS)

The System Status register is a read-only register that reflects the current state of the system.

Figure 19. System Status Register (SSCM_STATUS)

Offset:0x00 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	NXEN	PUB	SEC	0	BMODE			0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0/1	0/1	0/1	0	0	0	0	0

Table 21. SSCM_STATUS allowed register accesses

Access type	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 22. SSCM_STATUS field descriptions

Field	Description
NXEN	Nexus enabled
PUB	Public Serial Access Status. This bit indicates whether serial boot mode with public password is allowed. 1Serial boot mode with public password is allowed 0Serial boot mode with private flash memory password is allowed
SEC	Security Status. This bit reflects the current security state of the flash memory. 1The flash memory is secured. 0The flash memory is not secured.
BMODE	Device Boot Mode 000 Reserved 001 FlexCAN_0 Serial Boot Loader 010 LINFlex_0 Serial Boot Loader 011 Single Chip 100 Reserved 101 Reserved 110 Reserved 111 Reserved This field is only updated during reset.

System Memory Configuration Register (SSCM_MEMCONFIG)

The System Memory Configuration register is a read-only register that reflects the memory configuration of the system.

Offset: 0x02 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	PRSZ					PVLB	DTSZ				DVLD
W																
Reset	x	x	x	x	x	x	x	x	x	x	1	x	x	x	x	1

Figure 20. System Memory Configuration Register (SSCM_MEMCONFIG)

Table 23. SSCM_MEMCONFIG field descriptions

Field	Description
PRSZ	Code Flash Size 10000 128 KB 10001 256 KB 10010 384 KB 10011 512 KB
PVLB	Code Flash Available This bit identifies whether or not the on-chip code Flash is available in the system memory map. The Flash may not be accessible due to security limitations, or because there is no Flash in the system. 1 Code Flash is available 0 Code Flash is not available
DTSZ	Data Flash Size 0000 No Data Flash 0011 64 KB
DVLD	Data Flash Valid This bit identifies whether or not the on-chip Data Flash is visible in the system memory map. The Flash may not be accessible due to security limitations, or because there is no Flash in the system. 1 Data Flash is visible 0 Data Flash is not visible

Table 24. SSCM_MEMCONFIG allowed register accesses

Access type	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed (also reads SSCM_STATUS register)
Write	Not allowed	Not allowed	Not allowed

Error Configuration (SSCM_ERROR)

The Error Configuration register is a read-write register that controls the error handling of the system.

Offset: 0x06 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	PAE	RAE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21. Error Configuration (SSCM_ERROR)

Table 25. SSCM_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
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 Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (that is, at the PBRIDGE level). In this case, bits PAE and RAE will have no effect on the abort.

Table 26. SSCM_ERROR allowed register accesses

Access type	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed
Write	Allowed	Allowed	Not allowed

Debug Status Port Register (SSCM_DEBUGPORT)

The Debug Status Port register is used to (optionally) provide debug data on a set of pins.

Offset: 0x08 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	DEBUG_MODE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22. Debug Status Port Register (SSCM_DEBUGPORT)

Table 27. SSCM_DEBUGPORT field descriptions

Field	Description
DEBUG_MODE	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 28* describes the functionality of the Debug Status Port in each mode.

Table 28. Debug status port modes

Pin (1)	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
0	SSCM_STATUS [0]	SSCM_STATUS [8]	SSCM_MEMCONFIG [0]	SSCM_MEMCONFIG [8]	Reserved	Reserved	Reserved
1	SSCM_STATUS [1]	SSCM_STATUS [9]	SSCM_MEMCONFIG [1]	SSCM_MEMCONFIG [9]	Reserved	Reserved	Reserved
2	SSCM_STATUS [2]	SSCM_STATUS [10]	SSCM_MEMCONFIG [2]	SSCM_MEMCONFIG [10]	Reserved	Reserved	Reserved
3	SSCM_STATUS [3]	SSCM_STATUS [11]	SSCM_MEMCONFIG [3]	SSCM_MEMCONFIG [11]	Reserved	Reserved	Reserved
4	SSCM_STATUS [4]	SSCM_STATUS [12]	SSCM_MEMCONFIG [4]	SSCM_MEMCONFIG [12]	Reserved	Reserved	Reserved
5	SSCM_STATUS [5]	SSCM_STATUS [13]	SSCM_MEMCONFIG [5]	SSCM_MEMCONFIG [13]	Reserved	Reserved	Reserved
6	SSCM_STATUS [6]	SSCM_STATUS [14]	SSCM_MEMCONFIG [6]	SSCM_MEMCONFIG [14]	Reserved	Reserved	Reserved
7	SSCM_STATUS [7]	SSCM_STATUS [15]	SSCM_MEMCONFIG [7]	SSCM_MEMCONFIG [15]	Reserved	Reserved	Reserved

1. All signals are active high, unless otherwise noted

PIN[0..7] referred to in *Table 28* equates to PC[2..9] (Pad 34..41).

Table 29. SSCM_DEBUGPORT allowed register accesses

Access type	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 provide a means for the BAM code to unsecure the device via the SSCM if the password has been provided via serial download.

Figure 23. Password Comparison Register High Word (SSCM_PWCMPH)

Offset: 0x0C 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	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 24. Password Comparison Register Low Word (SSCM_PWCMPL)

Offset: 0x10 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	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 30. Password Comparison Register field descriptions

Field	Description
PWD_HI	Upper 32 bits of the password
PWD_LO	Lower 32 bits of the password

Table 31. SSCM_PWCMPH/L allowed register accesses

Access type	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).

In order to unsecure the device, the password needs to be written as follows: first the upper word to the SSCM_PWCMPH register, then the lower word to the SSCM_PWCMPL register. The SSCM compares the 64-bit password entered into the SSCM_PWCMPH / SSCM_PWCMPL registers with the NVPWM[1,0] private password stored in the shadow flash. If the passwords match then the SSCM temporarily uncensors the microcontroller.

6 Clock Description

This chapter describes the clock architectural implementation for SPC560Bx and SPC560Cx.

6.1 Clock architecture

System clocks are generated from three sources:

- Fast external crystal oscillator 4-16 MHz (FXOSC)
- Fast internal RC oscillator 16 MHz (FIRC)
- Frequency modulated phase locked loop (FMPLL)

Additionally, there are two low power oscillators:

- Slow internal RC oscillator 128 kHz (SIRC)
- Slow external crystal oscillator 32 KHz (SXOSC)

The clock architecture is shown in *Figure 25*.

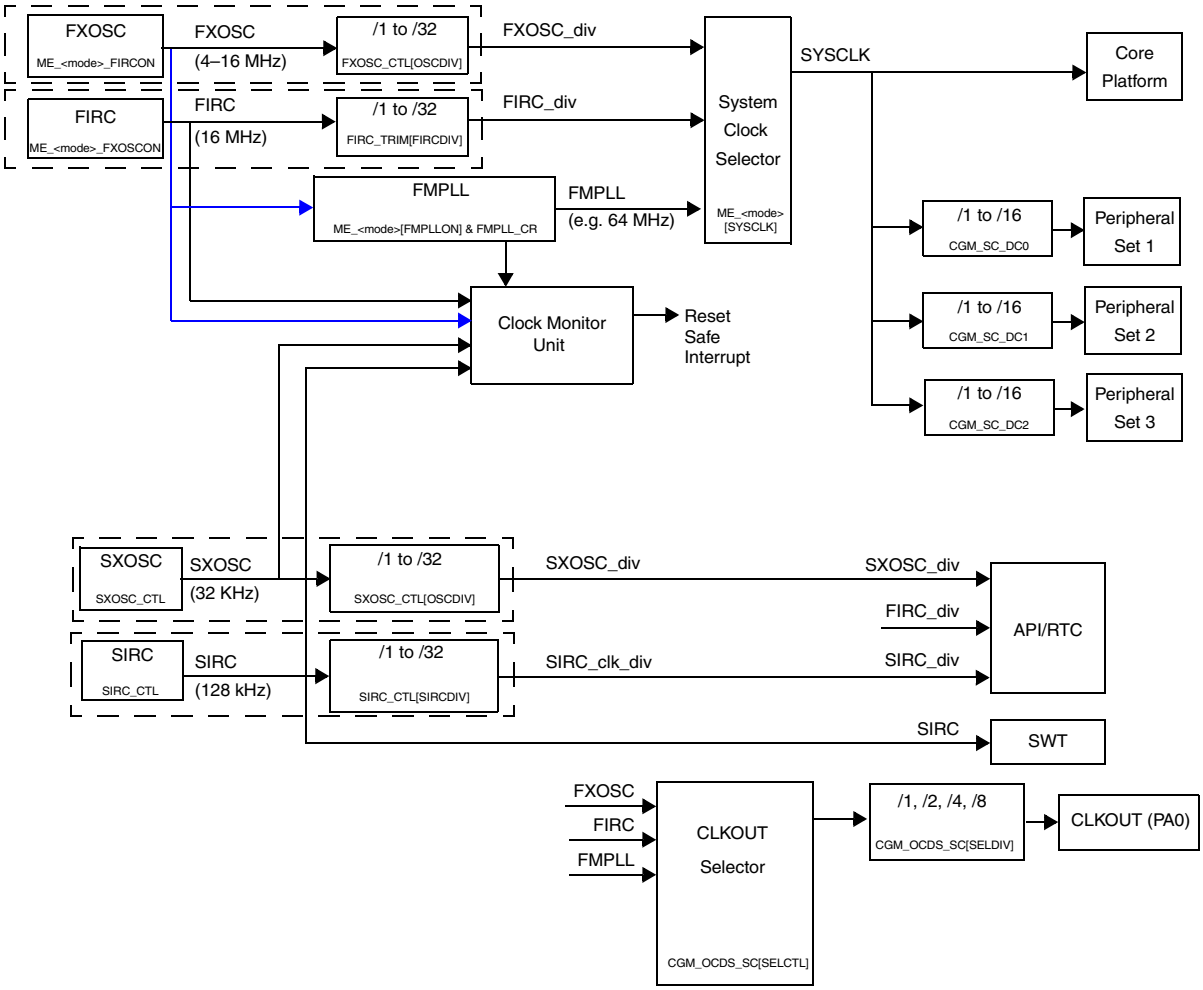


Figure 25. SPC560Bx and SPC560Cx system clock generation

6.2 Clock gating

The SPC560Bx and SPC560Cx provides the user with the possibility of gating the clock to the peripherals. *Table 32* describes for each peripheral the associated gating register address. See the ME_PCTLn section in this reference manual.

Additionally, peripheral set (1, 2 or 3) frequency can be configured to be an integer (1 to 16) divided version of the main system clock. See the CGM_SC_DC0 section in this reference manual for details.

Table 32. SPC560Bx and SPC560Cx — Peripheral clock sources

Peripheral	Register gating address offset (base = 0xC3FDC0C0)(1)	Peripheral set(2)
RPP_Z0H Platform	none (managed through ME mode)	—
DSPI_n	4+n (n = 0..2)	2
FlexCAN_n	16+n (n = 0..5)	2
ADC	32	3
I^2C	44	1
LINFLEX_n	48+n(n = 0..3)	1
CTU	57	3
CANS	60	—
SIUL	68	—
WKUP	69	—
eMIOS_n	72+n (n = 0..1)	3
RTC/API	91	—
PIT	92	—
CMU	104	—

1. See the ME_PCTL section in this reference manual for details.
2. "—" means undivided system clock.

6.3 Fast external crystal oscillator (FXOSC) digital interface

The FXOSC digital interface controls the operation of the 4–16 MHz fast external crystal oscillator (FXOSC). It holds control and status registers accessible for application.

6.3.1 Main features

- Oscillator powerdown control and status reporting through MC_ME block
- Oscillator clock available interrupt
- Oscillator bypass mode
- Output clock division factors ranging from 1, 2, 3....32

6.3.2 Functional description

The FXOSC circuit includes an internal oscillator driver and an external crystal circuitry. It provides an output clock that can be provided to the FMPLL or used as a reference clock to specific modules depending on system needs.

The FXOSC can be controlled by the MC_ME module. The ME_xxx_MC[FXOSCON] bit controls the powerdown of the oscillator based on the current device mode while ME_GS[S_XOSC] register provides the oscillator clock available status.

After system reset, the oscillator is put into powerdown state and software has to switch on when required. Whenever the crystal oscillator is switched on from the off state, the OSCCNT counter starts and when it reaches the value EOCV[7:0]×512, the oscillator clock

is made available to the system. Also, an interrupt pending FXOSC_CTL[I_OSC] bit is set. An interrupt is generated if the interrupt mask bit M_OSC is set.

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

Table 33 shows the truth table of different oscillator configurations.

Table 33. Truth table of crystal oscillator

ME_xxx_MC[FXOSCON]	FXOSC_CTL[OSCBYP]	XTAL	EXTAL	FXOSC	Oscillator mode
0	0	No crystal, High Z	No crystal, High Z	0	Powerdown, IDDQ
x	1	x	Ext clock	EXTAL	Bypass, OSC disabled
1	0	Crystal	Crystal	EXTAL	Normal, OSC enabled
		Gnd	Ext clock	EXTAL	Normal, OSC enabled

The FXOSC clock can be further divided by a configurable factor in the range 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by FXOSC_CTL[OSCDIV] field.

6.3.3 Register description

Figure 26. Fast External Crystal Oscillator Control Register (FXOSC_CTL)

Address: 0xC3FE_0000 Access: Special read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OSCBYP(1)	0	0	0	0	0	0	0	EOCV							
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	OSCDIV					I_OSC(2)	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

1. You can read this field, and you can write a value of "1" to it. Writing a "0" has no effect. A reset will also clear this bit.
2. You can write a value of "0" or "1" to this field. However, writing a "1" will clear this field, and writing "0" will have no effect on the field value.

Table 34. FXOSC_CTL field descriptions

Field	Description
OSCBYP	Crystal Oscillator bypass. This bit specifies whether the oscillator should be bypassed or not. 0 Oscillator output is used as root clock 1 EXTAL is used as root clock
EOCV	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 (OSCCNT runs on the FXOSC). 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 × 512, the crystal oscillator clock interrupt (I_OSC) request is generated. 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 is masked. 1 Crystal oscillator clock interrupt is enabled.
OSCDIV	Crystal oscillator clock division factor. This field specifies the crystal oscillator output clock division factor. The output clock is divided by the factor OSCDIV+1.
I_OSC	Crystal oscillator clock interrupt. This bit is set by hardware when OSCCNT counter reaches the count value EOCV × 512. 0 No oscillator clock interrupt occurred. 1 Oscillator clock interrupt pending.

6.4 Slow external crystal oscillator (SXOSC) digital interface

6.4.1 Introduction

The SXOSC digital interface controls the operation of the 32 KHz slow external crystal oscillator (SXOSC). It holds control and status registers accessible for application.

6.4.2 Main features

- Oscillator powerdown control and status
- Oscillator bypass mode
- Output clock division factors ranging from 1 to 32

6.4.3 Functional description

The SXOSC circuit includes an internal oscillator driver and an external crystal circuitry. It can be used as a reference clock to specific modules depending on system needs.

The SXOSC can be controlled via the SXOSC_CTL register. The OSCON bit controls the powerdown while bit S_OSC provides the oscillator clock available status.

After system reset, the oscillator is put to powerdown state and software has to switch on when required. Whenever the SXOSC is switched on from off state, the OSCCNT counter starts and when it reaches the value EOCV[7:0]×512, the oscillator clock is made available to the system.

The oscillator circuit can be bypassed by writing SXOSC_CTL[OSCBYP] bit to '1'. This bit can only be set by software. A system reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as the external clock applied on the OSC32K_EXTAL pin and the oscillator status is forced to '1'. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 35 shows the truth table of different configurations of the oscillator.

Table 35. SXOSC truth table

SXOSC_CTL fields		OSC32K_XTAL	OSC32K_EXTAL	SXOSC	Oscillator MODE
OSCON	OSCBYP				
0	0	No crystal, High Z	No crystal, High Z	0	Powerdown, IDDQ
x	1	x	External clock	OSC32K_EXTAL	Bypass, OSC disabled
1	0	Crystal	Crystal	OSC32K_EXTAL	Normal, OSC enabled
		Ground	External clock	OSC32K_EXTAL	Normal, OSC enabled

The SXOSC clock can be further divided by a configurable factor in the range 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by SXOSC_CTL[OSCDIV] field.

6.4.4 Register description

Figure 27. Slow External Crystal Oscillator Control Register (SXOSC_CTL)

Address: 0xC3FE_0040 Access: Special read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OSCBYP(1)	0	0	0	0	0	0	0	EOCV							
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	0	0	0	OSCDIV					0	0	0	0	0	0	S_OSC	OSCON
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. You can read this field, and you can write a value of "1" to it. Writing a "0" has no effect. A reset will also clear this bit.

Table 36. SXOSC_CTL field descriptions

Field	Description
OSCBYP	Crystal Oscillator bypass. This bit specifies whether the oscillator should be bypassed or not. 0 Oscillator output is used as root clock. 1 OSC32K_EXTAL is used as root clock.
EOCV	End of Count Value. This field specifies 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 × 512, the crystal oscillator status (S_OSC) is set. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
OSCDIV	Crystal oscillator clock division factor. This field specifies the crystal oscillator output clock division factor. The output clock is divided by the factor OSCDIV + 1.
S_OSC	Crystal oscillator status. 0 Crystal oscillator output clock is not stable. 1 Crystal oscillator is providing a stable clock.
OSCON	Crystal oscillator enable. 0 Crystal oscillator is switched off. 1 Crystal oscillator is switched on.

Note: The 32 KHz slow external crystal oscillator is by default always ON, but can be configured OFF in standby by setting the OSCON bit.

6.5 Slow internal RC oscillator (SIRC) digital interface

6.5.1 Introduction

The SIRC digital interface controls the 128 kHz slow internal RC oscillator (SIRC). It holds control and status registers accessible for application.

6.5.2 Functional description

The SIRC provides a low frequency (f_{SIRC}) clock of 128 kHz requiring very low current consumption. This clock can be used as the reference clock when a fixed base time is required for specific modules.

SIRC is always on in all device modes except STANDBY mode. In STANDBY mode, it is controlled by SIRC_CTL[SIRCON_STDBY] bit. The clock source status is updated in SIRC_CTL[S_SIRC] bit.

The SIRC clock can be further divided by a configurable division factor in the range from 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by SIRC_CTL[SIRCDIV] bits.

The SIRC output frequency can be trimmed using SIRC_CTL[SIRCTRIM]. After a power-on reset, the SIRC 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 SIRC_CTL[SIRCTRIM]

and this field shows a value of zero. Therefore, be aware that the SIRC_CTL[SIRCTRIM] does not reflect the current trim value until you have written to this field. Pay particular attention to this feature when you initiate a read-modify-write operation on SIRC_CTL, because a SIRCTRIM value of zero may be unintentionally written back and this may alter the SIRC frequency. In this case, you should calibrate the SIRC using the CMU or be sure that you only write to the upper 16 bits of this SIRC_CTL.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from –16 to 15. 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.

6.5.3 Register description

Figure 28. Low Power RC Control Register (SIRC_CTL)

Address: 0xC3FE_0080 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	SIRCTRIM				
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	SIRCDIV					0	0	0	S_SIRC	0	0	0	SIRCON_STDBY
W																
RESET:	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Table 37. SIRC_CTL field descriptions

Field	Description
SIRCTRIM	SIRC trimming bits. This field corresponds (via two's complement) to a trim factor of –16 to +15. A +1 change in SIRCTRIM decreases the current frequency by $\Delta_{SIRCTRIM}$ (see the device data sheet). A –1 change in SIRCTRIM increases the current frequency by $\Delta_{SIRCTRIM}$ (see the device data sheet).
SIRCDIV	SIRC clock division factor. This field specifies the SIRC oscillator output clock division factor. The output clock is divided by the factor SIRCDIV+1.

Table 37. SIRC_CTL field descriptions (continued)

Field	Description
S_SIRC	SIRC clock status. 0 SIRC is not providing a stable clock. 1 SIRC is providing a stable clock.
SIRCON_STDBY	SIRC control in STANDBY mode. 0 SIRC is switched off in STANDBY mode. 1 SIRC is switched on in STANDBY mode.

6.6 Fast internal RC oscillator (FIRC) digital interface

6.6.1 Introduction

The FIRC digital interface controls the 16 MHz fast internal RC oscillator (FIRC). It holds control and status registers accessible for application.

6.6.2 Functional description

The FIRC provides a high frequency (f_{FIRC}) clock of 16 MHz. This clock can be used to accelerate the exit from reset and wakeup sequence from low power modes of the system. It is controlled by the MC_ME module based on the current device mode. The clock source status is updated in ME_GS[S_RC]. Please refer to the MC_ME chapter for further details.

The FIRC can be further divided by a configurable division factor in the range from 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by RC_CTL[RCDIV] bits.

The FIRC output frequency can be trimmed using FIRC_CTL[FIRCTRIM]. After a power-on reset, the FIRC 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 FIRC_CTL[FIRCTRIM], and this field will show a value of zero. Therefore, be aware that the FIRC_CTL[FIRCTRIM] 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 FIRC_CTL, because a FIRCTRIM value of zero may be unintentionally written back and this may alter the FIRC frequency. In this case, you should calibrate the FIRC using the CMU or ensure that you write only to the upper 16 bits of this FIRC_CTL.

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.

During STANDBY mode entry process, the FIRC is controlled based on ME_STANDBY_MC[RCON] bit. This is the last step in the standby entry sequence. On any system wake-up event, the device exits STANDBY mode and switches on the FIRC. The actual powerdown status of the FIRC when the device is in standby is provided by RC_CTL[FIRCON_STDBY] bit.

6.6.3 Register description

Figure 29. FIRC Oscillator Control Register (FIRC_CTL)

Address: 0xC3FE_0060 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	FIRCTRIM					
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	FIRCDIV					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 38. FIRC_CTL field descriptions

Field	Description
FIRCTRIM	FIRC trimming bits. This field corresponds (via two's complement) to a trim factor of –16 to +15. A +1 change in FIRCTRIM decreases the current frequency by $\Delta_{FIRCTRIM}$ (see the device data sheet). A –1 change in SIRCTRIM increases the current frequency by $\Delta_{FIRCTRIM}$ (see the device data sheet).
FIRCDIV	FIRC clock division factor. This field specifies the FIRC oscillator output clock division factor. The output clock is divided by the factor FIRCDIV+1.

6.7 Frequency-modulated phase-locked loop (FMPLL)

6.7.1 Introduction

This section describes the features and functions of the FMPLL module implemented in the device.

6.7.2 Overview

The FMPLL enables the generation of high speed system clocks from a common 4–16 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor and output clock divider ratio are all software configurable.

SPC560Bx and SPC560Cx has one FMPLL that can generate the system clock and takes advantage of the FM mode.

Note: The user must take care not to program device with a frequency higher than allowed (no hardware check).

The FMPLL block diagram is shown in *Figure 30*.

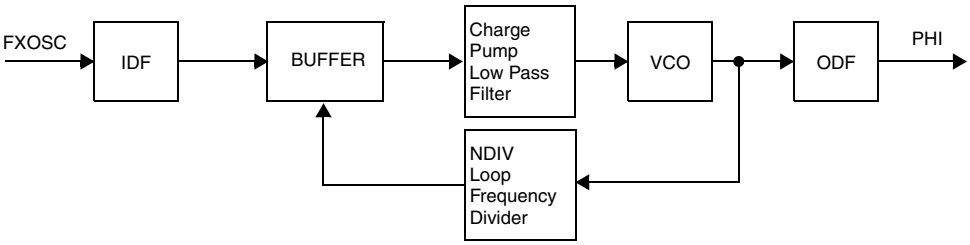


Figure 30. FMPLL block diagram

6.7.3 Features

The FMPLL has the following major features:

- Input clock frequency 4 MHz – 16 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Frequency divider (FD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated FMPLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth
  - ±0.25% to ±4% deviation from center spread frequency[e]
  - –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 frequency modulation
  - Powerdown mode

6.7.4 Memory map[f]

Table 39 shows the memory map of the FMPLL.

e. Spread spectrum should be programmed in line with maximum datasheet frequency figures.

f. FMPLL_x are mapped through the ME_CGM register slot

Table 39. FMPLL memory map

Base address: 0xC3FE_00A0		
Address offset	**Register**	**Location**
0x0	*Control Register (CR)*	*on page 6-117*
0x4	Modulation Register (MR)	*on page 6-119*

6.7.5 Register description

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

Control Register (CR)

Figure 31. Control Register (CR)

Offset: 0x0 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	IDF				ODF		0	NDIV						
W																
Reset	0	0	0	0	0	0	1	0	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 40. CR field descriptions

Field	Description
IDF	The value of this field sets the FMPLL input division factor as described in *Table 41.*
ODF	The value of this field sets the FMPLL output division factor as described in *Table 42.*
NDIV	The value of this field sets the FMPLL loop division factor as described in *Table 43.*
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. 0 Progressive clock switching disabled. 1 Progressive clock switching enabled. *Note: Note: Progressive clock switching should not be used if a non-changing clock is needed, such as for serial communications, until the division has finished.*

Table 40. CR field descriptions (continued)

Field	Description
UNLOCK_ONCE	This bit is a sticking indication of FMPLL loss of lock condition. UNLOCK_ONCE is set when the FMPLL loses lock. Whenever the FMPLL reacquires lock, UNLOCK_ONCE remains set. Only a power-on reset clears this bit.
I_LOCK	This bit is set by hardware whenever there is a lock/unlock event.
S_LOCK	This bit is an indication of whether the FMPLL has acquired lock. 0: FMPLL unlocked 1: FMPLL locked *Note:*
PLL_FAIL_MASK	This bit is used to mask 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 FMPLL is switched on. It is cleared by software writing ‘1’.

Table 41. Input divide ratios

IDF[3:0]	Input divide ratios
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

Table 42. Output divide ratios

ODF[1:0]	Output divide ratios
00	Divide by 2
01	Divide by 4
10	Divide by 8
11	Divide by 16

Table 43. Loop divide ratios

NDIV[6:0]	Loop divide ratios
0000000–0011111	—
0100000	Divide by 32
0100001	Divide by 33
0100010	Divide by 34
...	...
1011111	Divide by 95
1100000	Divide by 96
1100001–1111111	—

Modulation Register (MR)

Figure 32. Modulation Register (MR)

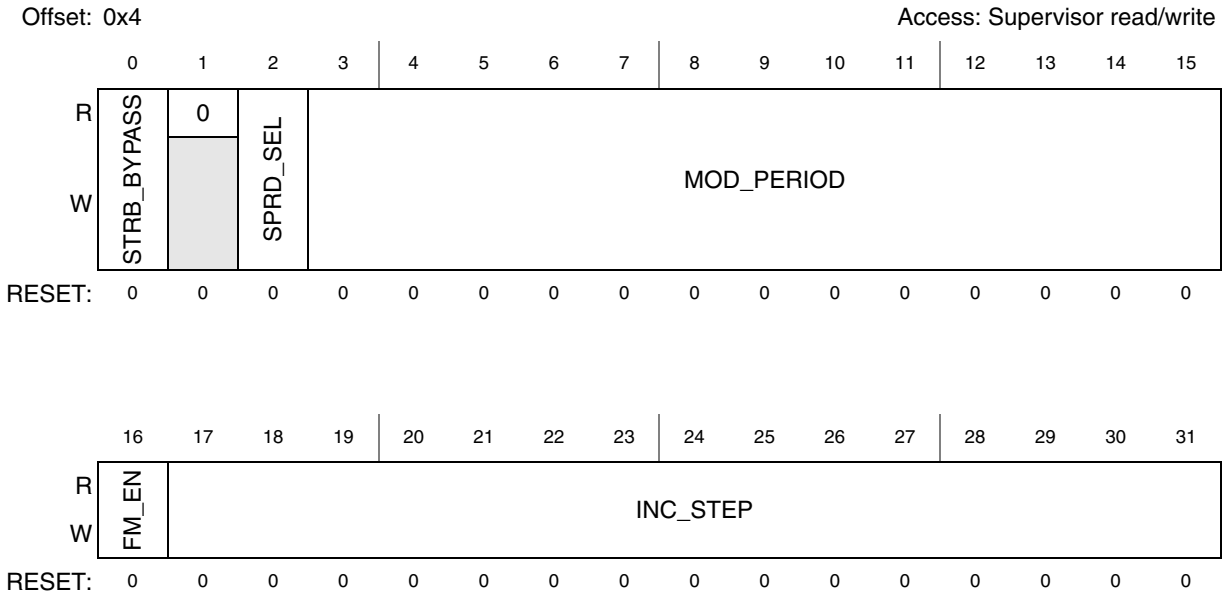


Table 44. MR field descriptions

Field	Description
STRB_BYPASS	Strobe bypass. The STRB_BYPASS signal is used to bypass the strobe signal used inside FMPLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0 Strobe is used to latch FMPLL modulation control bits 1 Strobe is bypassed. In this case control bits need to be static. The control bits must be changed only when FMPLL is in powerdown mode.
SPRD_SEL	Spread type selection. The SPRD_SEL controls the spread type in Frequency Modulation mode. 0 Center SPREAD 1 Down SPREAD
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: f_{ref}: represents the frequency of the feedback divider f_{mod}: represents the modulation frequency
FM_EN	Frequency Modulation Enable. The FM_EN 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 (CR[NDIV])

6.7.6 Functional description

Normal mode

In Normal Mode the FMPLL inputs are driven by the CR. This means that, when the FMPLL is in lock state, the FMPLL output clock (PHI) is derived by the reference clock (XOSC) through this relation:

$$phi = \frac{clkin \cdot NDIV}{IDF \cdot ODF}$$

where the value of IDF, NDIV and ODF are set in the CR and can be derived from *Table 41*, *Table 42* and *Table 43*.

Table 45. FMPLL lookup table

Crystal frequency (MHz)	FMPLL output frequency (MHz)	CR field values			VCO frequency (MHz)
		IDF	ODF	NDIV	
8	32	0	2	32	256
	64	0	2	64	512
	80	0	1	40	320
16	32	1	2	32	256
	64	1	2	64	512
	80	1	1	40	320
40	32	4	2	32	256
	64	4	2	64	512
	80	3	1	32	320

Progressive clock switching

Progressive clock switching allows to switch the system clock to FMPLL 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 CR[EN_PLL_SW] bit. When enabled, the system clock is switched to divided PHI. The FMPLL_clk divider is then progressively decreased to the target divider as shown in *Table 46*.

Table 46. Progressive clock switching on pll_select rising edge

Number of FMPLL output clock cycles	FMPLL_clk frequency (FMPLL output clock frequency)
8	(FMPLL output clock frequency)/8
16	(FMPLL output clock frequency)/4
32	(FMPLL output clock frequency)/2
onward	FMPLL output clock frequency

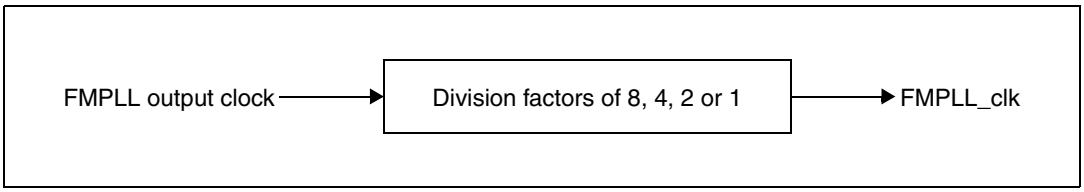


Figure 33. FMPLL output clock division flow during progressive switching

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.

FM mode is activated in two steps:

1. Configure the FM mode characteristics: MOD_PERIOD, INC_STEP.
2. Enable the FM mode by programming bit FM_EN of the MR to '1'. FM mode can only be enabled when FMPLL 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 FMPLL only when the strobe signal goes high for at least two cycles of CLKIN clock. The strobe signal is automatically generated in the FMPLL digital interface when the modulation is enabled (FM_EN goes high) if the FMPLL is locked (S_LOCK = 1) or when the modulation has been enabled (FM_EN = 1) and FMPLL enters 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) need to be static or hardwired to constant values. The control bits must be changed only when the FMPLL is in powerdown mode.

The modulation depth in % is

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

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

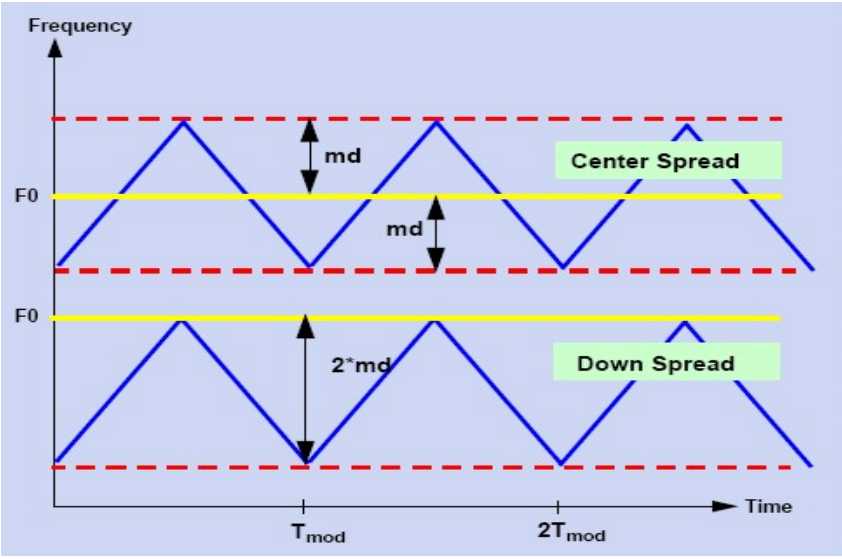


Figure 34. Frequency modulation

Powerdown mode

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

6.7.7 Recommendations

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

- The FMPLL 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 FMPLL output clock is not selected as system clock. Use progressive clock switching if system clock changes are required while the PLL is being used as the system clock source. MOD_PERIOD, INC_STEP, SPREAD_SEL bits should be modified before activating the FM 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 FMPLL is in powerdown mode.
- Use progressive clock switching (FMPLL output clock can be changed when it is the system clock, but only when using progressive clock switching).

6.8 Clock monitor unit (CMU)

6.8.1 Introduction

The Clock Monitor Unit (CMU), also referred to as Clock Quality Checker or Clock Fault Detector, serves two purposes. The main task is to permanently supervise the integrity of the various clock sources, for example a crystal oscillator or FMPLL. In case the FMPLL leaves an upper or lower frequency boundary or the crystal oscillator fails it can detect and forward these kind of events towards the MC_ME and MC_CGM. The clock management unit in turn can then switch to a SAFE mode where it uses the default safe clock source (FIRC), reset the device or generate the interrupt according to the system needs.

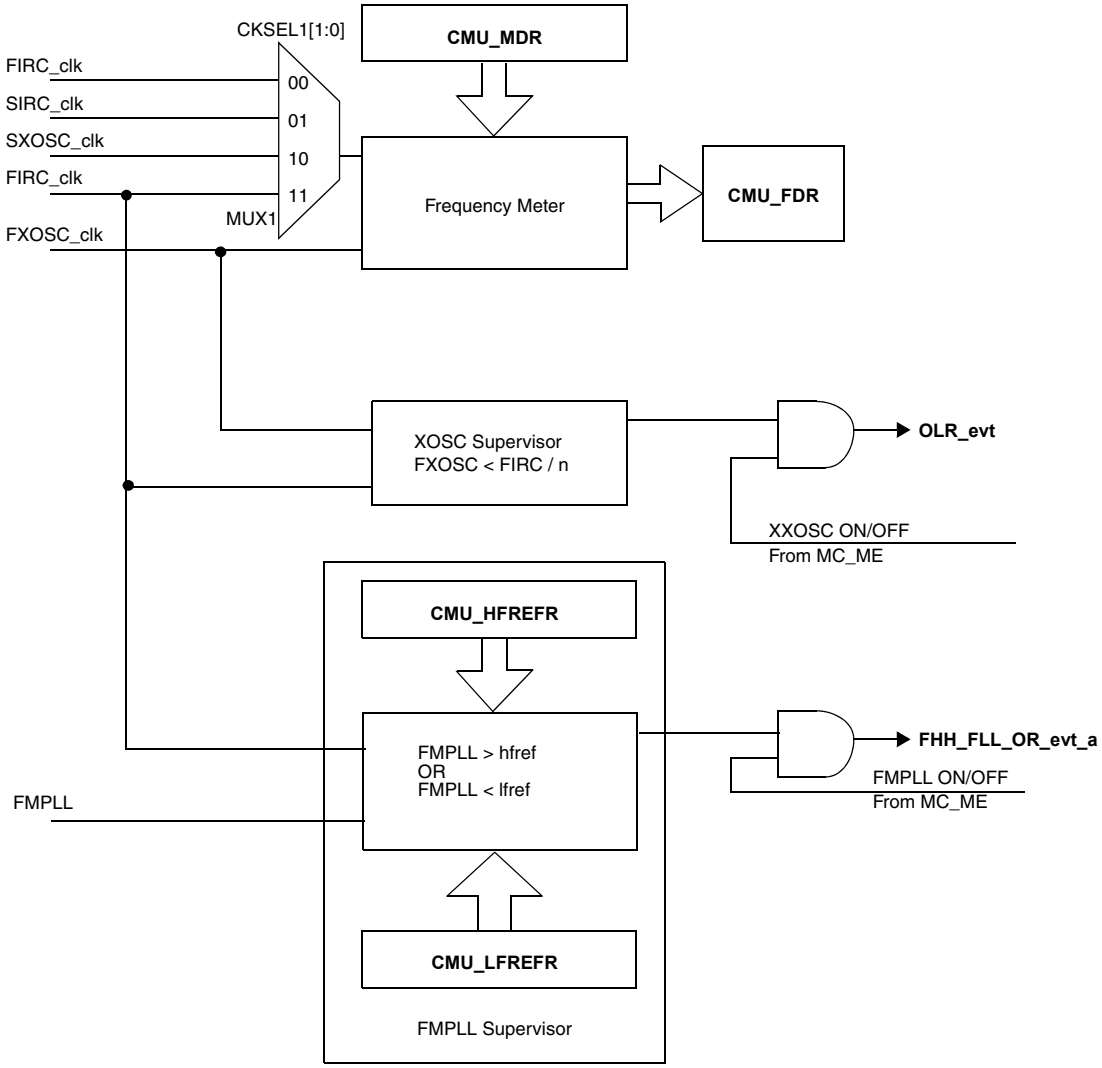
It can also monitor the external crystal oscillator clock, which must be greater than the internal RC clock divided by a division factor given by CMU_CSR[RCDIV], and generates a system clock transition request or an interrupt when enabled.

The second task of the CMU is to provide a frequency meter, which allows to measure the frequency of one clock source vs. a reference clock. This is useful to allow the calibration of the on-chip RC oscillator(s), as well as being able to correct/calculate the time deviation of a counter which is clocked by the RC oscillator.

6.8.2 Main features

- FIRC, SIRC, SXOSC oscillator frequency measurement using FXOSC as reference clock
- External oscillator clock monitoring with respect to FIRC_clk/n clock
- FMPLL clock frequency monitoring for a high and low frequency range with FIRC as reference clock
- Event generation for various failures detected inside monitoring unit

6.8.3 Block diagram



OLR_evt : It is the event signalling XOSC failure when asserted. When this signal is asserted, RGM may generate reset, interrupt or SAFE request based on the RGM configuration.

FHH_FLL_OR_evt_a : It is the event signalling FMPLL failure when asserted. Based on the CMU_HFREFR and CMU_LFREFR configuration, if the FMPLL is greater than hign frequency range or less than the low frequency range configuration, this signal is generated. When this signal is asserted, RGM may generate reset, interrupt or SAFE request based on the RGM configuration.

Figure 35. Clock Monitor Unit diagram

6.8.4 Functional description

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

- FXOSC_clk: clock coming from the fast external crystal oscillator
- SXOSC_clk: clock coming from the slow external crystal oscillator
- SIRC_clk: clock coming from the slow (low frequency) internal RC oscillator
- FIRC_clk: clock coming from the fast (high frequency) internal RC oscillator
- FMPLL_clk: clock coming from the FMPLL
- $f_{FXOSC\_clk:}$ frequency of fast external crystal oscillator clock
- f_{SXOSC_clk}: frequency of slow external crystal oscillator clock
- f_{SIRC_clk}: frequency of slow (low frequency) internal RC oscillator
- f_{FIRC_clk}: frequency of fast (high frequency) internal RC oscillator
- f_{FMPLL_clk}: frequency of FMPLL clock

Crystal clock monitor

If f_{FXOSC_clk} is less than f_{FIRC_clk} divided by 2^{RCDIV} bits of the CMU_CSR and the FXOSC_clk is ‘ON’ as signalled by the MC_ME then:

- An event pending bit OLRI in CMU_ISR is set.
- A failure event OLR is signalled to the MC_RGM which in turn can automatically switch to a safe fallback clock and generate an interrupt or reset.

FMPLL clock monitor

The f_{FMPLL_clk} can be monitored by programming bit CME of the CMU_CSR register to ‘1’. The FMPLL_clk monitor starts as soon as bit CME is set. This monitor can be disabled at any time by writing bit CME to ‘0’.

If f_{FMPLL_clk} is greater than a reference value determined by bits HFREF[11:0] of the CMU_HFREFR and the FMPLL_clk is ‘ON’, as signalled by the MC_ME, then:

- An event pending bit FHHI in CMU_ISR is set.
- A failure event is signalled to the MC_RGM which in turn can generate an interrupt or safe mode request or functional reset depending on the programming model.

If f_{FMPLL_clk} is less than a reference value determined by bits LFREF[11:0] of the CMU_LFREFR and the FMPLL_clk is ‘ON’, as signaled by the MC_ME, then:

- An event pending bit FLLI in CMU_ISR is set.
- A failure event FLL is signalled to the MC_RGM which in turn can generate an interrupt or safe mode request or functional reset depending on the programming model.

Note: The internal RC oscillator is used as reliable reference clock for the clock supervision. In order to avoid false events, proper programming of the dividers is required. These have to take into account the accuracy and frequency deviation of the internal RC oscillator.

Note: If PLL frequency goes out of range, the CMU shall generate FMPLL fll/fhh event. It takes approximately 5 µs to generate this event.

Frequency meter

The purpose of the frequency meter is twofold:

- to measure the frequency of the oscillators SIRC, FIRC or SXOSC
- to calibrate an internal RC oscillator (SIRC or FIRC) using a known frequency

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

The reference clock is always the FXOSC_clk. The frequency meter returns a precise value of frequencies f_{SXOSC_clk}, f_{FIRC_clk} or f_{SIRC_clk} according to CKSEL1 bit value. 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 clock cycles of the selected clock source 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_x[g] can be derived from the value loaded in the CMU_FDR as follows:

Equation 1 $f_x = (f_{FXOSC} \times MD) / n$

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

The frequency meter by default evaluates f_{FIRC_clk}, but software can swap to f_{SIRC_clk} or f_{SXOSC_clk} by programming the CKSEL bits in the CMU_CSR.

6.8.5 Memory map and register description

The memory map of the CMU is shown in *Table 47*.

Table 47. CMU memory map

Base address: 0xC3FE_0100			
Register name	**Address offset**	**Reset value**	**Location**
Control Status Register (CMU_CSR)	0x00	0x00000006	*on page 6-127*
Frequency Display Register (CMU_FDR)	0x04	0x00000000	*on page 6-128*
High Frequency Reference Register FMPLL (CMU_HFREFR)	0x08	0x00000FFF	*on page 6-128*
Low Frequency Reference Register FMPLL (CMU_LFREFR)	0x0C	0x00000000	*on page 6-129*
Interrupt Status Register (CMU_ISR)	0x10	0x00000000	*on page 6-129*
Reserved	0x14	0x00000000	—
Measurement Duration Register (CMU_MDR)	0x18	0x00000000	*on page 6-130*

g. x = FIRC,SIRC or SXOSC

Control Status Register (CMU_CSR)

Figure 36. Control Status Register (CMU_CSR)

Offset: 0x00 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	SFM(1)	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	CKSEL1		0	0	0	0	0	RCDIV		CME_A
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

1. You can read this field, and you can write a value of "1" to it. Writing a "0" has no effect. A reset will also clear this bit.

Table 48. CMU_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 register. 0 Frequency measurement completed or not yet started. 1 Frequency measurement not completed.
CKSEL1	Clock oscillator selection bit. CKSEL1 selects the clock to be measured by the frequency meter. 00 FIRC_clk selected. 01 SIRC_clk selected. 10 SXOSC_clk selected. 11 FIRC_clk selected.
RCDIV	RC clock division factor . These bits specify the RC clock division factor. The output clock is FIRC_clk divided by the factor 2^{RCDIV}. This output clock is used to compare with FXOSC_clk 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_A	FMPLL_0 clock monitor enable. 0 FMPLL_0 monitor disabled. 1 FMPLL_0 monitor enabled.

Frequency Display Register (CMU_FDR)

Figure 37. Frequency Display Register (CMU_FDR)

Offset: 0x04 Access: 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 49. CMU_FDR field descriptions

Field	Description
FD	Measured frequency bits. This register displays the measured frequency f_x with respect to f_{FXOSC}. The measured value is given by the following formula: $f_x = (f_{FXOSC} \times MD) / n$, where n is the value in CMU_FDR register. *Note: x = FIRC, SIRC or SXOSC.*

High Frequency Reference Register FMPLL (CMU_HFREFR)

Figure 38. High Frequency Reference Register FMPLL (CMU_HFREFR)

Offset: 0x08 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	HFREF											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 50. CMU_HFREFR field descriptions

Field	Description
HFREF	High Frequency reference value. This field determines the high reference value for the FMPLL clock. The reference value is given by: $(HFREF \div 16) \times (f_{FIRC} \div 4)$.

Low Frequency Reference Register FMPLL (CMU_LFREFR)

Figure 39. Low Frequency Reference Register FMPLL (CMU_LFREFR)

Offset: 0x0C 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	LFREF											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 51. CMU_LFREFR field descriptions

Field	Description
LFREF	Low Frequency reference value. This field determines the low reference value for the FMPLL. The reference value is given by: (LFREF ÷ 16) × (f_{FIRC} ÷ 4).

Interrupt Status Register (CMU_ISR)

Figure 40. Interrupt status register (CMU_ISR)

Offset: 0x10 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	FHHI	FLLI	OLRI
W														w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 52. CMU_ISR field descriptions

Field	Description
FHHI	FMPLL clock frequency higher than high reference interrupt. This bit is set by hardware when f_{FMPLL_clk} becomes higher than HFREF value and FMPLL_clk is 'ON' as signalled by the MC_ME. It can be cleared by software by writing '1'. 0 No FHH event. 1 FHH event is pending.

Table 52. CMU_ISR field descriptions (continued)

FLLI	FMPLL clock frequency lower than low reference event. This bit is set by hardware when f_{FMPLL_clk} becomes lower than LFREF value and FMPLL_clk is 'ON' as signalled by the MC_ME. It can be cleared by software by writing '1'. 0 No FLL event. 1 FLL event is pending.
OLRI	Oscillator frequency lower than RC frequency event. This bit is set by hardware when f_{FXOSC_clk} is lower than FIRC_clk/2^{RCDIV} frequency and FXOSC_clk is 'ON' as signalled by the MC_ME. It can be cleared by software by writing '1'. 0 No OLR event. 1 OLR event is pending.

Measurement Duration Register (CMU_MDR)

Figure 41. Measurement Duration Register (CMU_MDR)

Offset: 0x18 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	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 53. CMU_MDR field descriptions

Field	Description
MD	Measurement duration bits. This field displays the measurement duration in numbers of clock cycles of the selected clock source. This value is loaded in the frequency meter downcounter. When CMU_CSR[SFM] = 1, the downcounter starts counting.

7 Clock Generation Module (MC_CGM)

7.1 Overview

The clock generation module (MC_CGM) generates reference clocks for all SoC 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 utilized 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 42 depicts the MC_CGM block diagram.

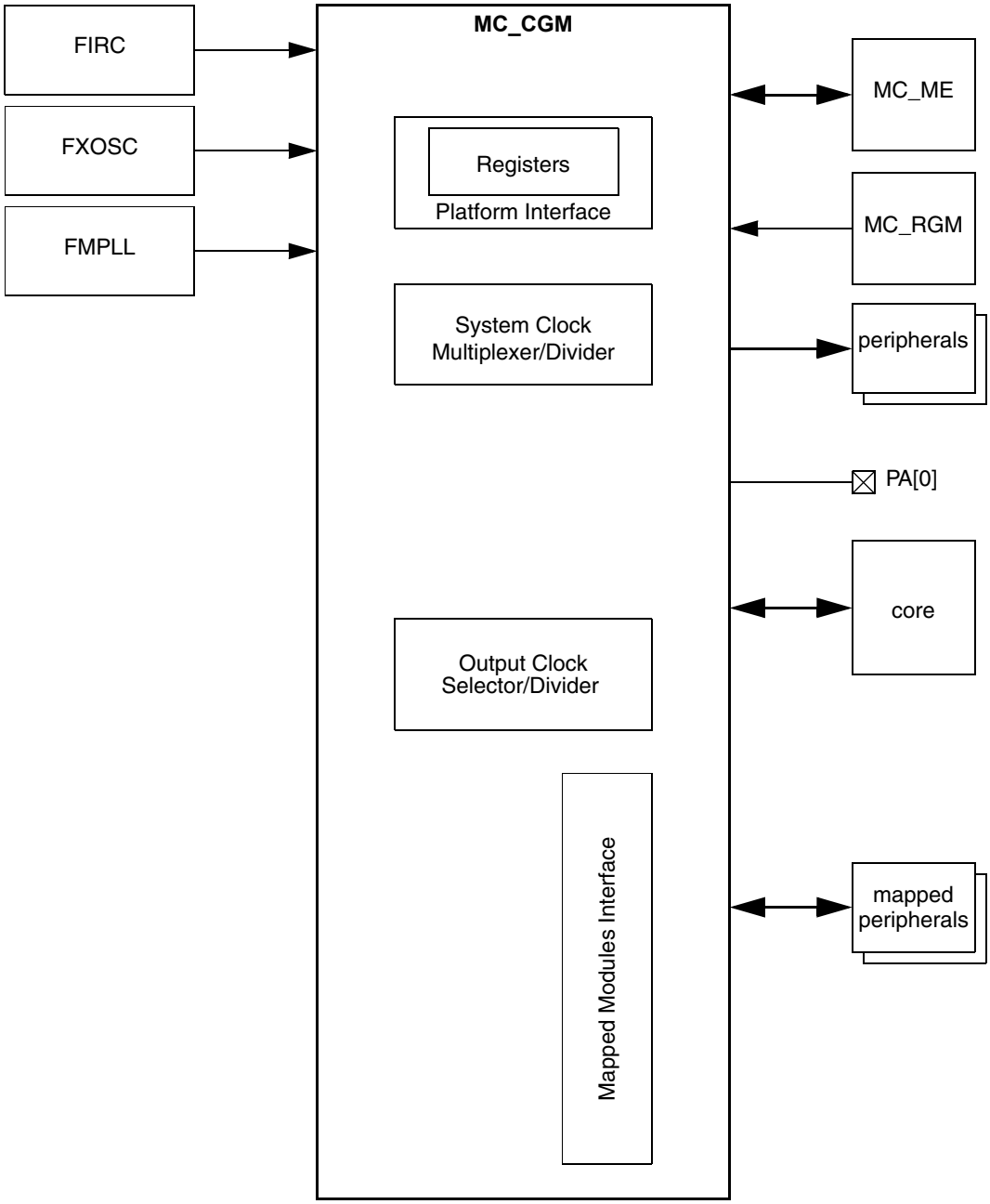


Figure 42. MC_CGM Block Diagram

7.2 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

7.3 Modes of Operation

This section describes the basic functional modes of the MC_CGM.

7.3.1 Normal and Reset Modes of Operation

During normal and reset modes of operation, the clock selection for the system clock is controlled by the MC_ME.

7.4 External Signal Description

The MC_CGM delivers an output clock to the PA[0] pin for off-chip use and/or observation.

7.5 Memory Map and Register Definition

Table 54. MC_CGM Register Description

Address	Name	Description	Size	Access	Location
				Supervisor	
0xC3FE_0370	CGM_OC_EN	Output Clock Enable	word	read/write	*on page 7-138*
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read/write	*on page 7-138*
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	*on page 7-139*
0xC3FE_037C	CGM_SC_DC0	System Clock Divider Configuration 0	byte	read/write	*on page 7-140*
0xC3FE_037D	CGM_SC_DC1	System Clock Divider Configuration 1	byte	read/write	*on page 7-140*
0xC3FE_037E	CGM_SC_DC2	System Clock Divider Configuration 2	byte	read/write	*on page 7-140*

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 55. MC_CGM Memory Map

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_0000 … 0xC3FE_001C	FXOSC registers																
0xC3FE_0020 … 0xC3FE_003C	reserved																
0xC3FE_0040 … 0xC3FE_005C	SXOSC registers																
0xC3FE_0060 … 0xC3FE_007C	FIRC registers																
0xC3FE_0080 … 0xC3FE_009C	SIRC registers																
0xC3FE_00A0 … 0xC3FE_00BC	FMPLL registers																
0xC3FE_00C0 … 0xC3FE_00DC	reserved																
0xC3FE_00E0 … 0xC3FE_00FC	reserved																
0xC3FE_0100 … 0xC3FE_011C	CMU registers																

Table 55. MC_CGM Memory Map (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_0120 … 0xC3FE_013C	reserved																
0xC3FE_0140 … 0xC3FE_015C	reserved																
0xC3FE_0160 … 0xC3FE_017C	reserved																
0xC3FE_0180 … 0xC3FE_019C	reserved																
0xC3FE_01A0 … 0xC3FE_01BC	reserved																
0xC3FE_01C0 … 0xC3FE_01DC	reserved																
0xC3FE_01E0 … 0xC3FE_01FC	reserved																
0xC3FE_0200 … 0xC3FE_021C	reserved																
0xC3FE_0220 … 0xC3FE_023C	reserved																

Table 55. MC_CGM Memory Map (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_0240 … 0xC3FE_025C	reserved																
0xC3FE_0260 … 0xC3FD_C27C	reserved																
0xC3FE_0280 … 0xC3FE_029C	reserved																
0xC3FE_02A0 … 0xC3FE_02BC	reserved																
0xC3FE_02C0 … 0xC3FE_02DC	reserved																
0xC3FE_02E0 … 0xC3FE_02FC	reserved																
0xC3FE_0300 … 0xC3FE_031C	reserved																
0xC3FE_0320 … 0xC3FE_033C	reserved																
0xC3FE_0340 … 0xC3FE_035C	reserved																

Table 55. MC_CGM Memory Map (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_0360 … 0xC3FE_036C	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	CGM_SC_DC0…2	R	DE0	0	0	0	DIV0				DE1	0	0	0	DIV1			
		W																
		R	DE2	0	0	0	DIV2				0	0	0	0	0	0	0	0
		W																
0xC3FE_0400 … 0xC3FE_3FFC	reserved																	

7.5.1 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 43. Output Clock Enable Register (CGM_OC_EN)

Address 0xC3FE_0370 Access: Supervisor 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 56. 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 44. Output Clock Division Select Register (CGM_OCDS_SC)

Address 0xC3FE_0374 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SELDIV		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 output clock source and by which factor it is divided before being delivered at the output clock.

Table 57. Output Clock Division Select Register (CGM_OCDS_SC) 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 4-16 MHz ext. xtal osc. 0001 16 MHz int. RC osc. 0010 freq. mod. PLL 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 45. System Clock Select Status Register (CGM_SC_SS)

Address 0xC3FE_0378 Access: Supervisor 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 clock source selection for the following clocks:

- undivided: system clock
- divided by system clock divider 0: peripheral set 1 clock
- divided by system clock divider 1: peripheral set 2 clock
- divided by system clock divider 2: peripheral set 3 clock

See *Figure 47* for details.

Table 58. 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 16 MHz int. RC osc. 0001 div. 16 MHz int. RC osc. 0010 4-16 MHz ext. xtal osc. 0011 div. ext. xtal osc. 0100 freq. mod. PLL 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

System Clock Divider Configuration Registers (CGM_SC_DC0...2)

Figure 46. System Clock Divider Configuration Registers (CGM_SC_DC0…2)

Address 0xC3FE_037C Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				DE1	0	0	0	DIV1			
W																
Reset	1	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	DE2	0	0	0	DIV2				0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

These registers control the system clock dividers.

Table 59. System Clock Divider Configuration Registers (CGM_SC_DC0…2) Field Descriptions

Field	Description
DE0	**Divider 0 Enable** 0 Disable system clock divider 0 1 Enable system clock divider 0
DIV0	**Divider 0 Division Value** — The resultant peripheral set 1 clock will have a period DIV0 + 1 times that of the system clock. If the DE0 is set to ‘0’ (Divider 0 is disabled), any write access to the DIV0 field is ignored and the peripheral set 1 clock remains disabled.
DE1	**Divider 1 Enable** 0 Disable system clock divider 1 1 Enable system clock divider 1

Table 59. System Clock Divider Configuration Registers (CGM_SC_DC0…2) Field Descriptions

Field	Description
DIV1	**Divider 1 Division Value** — The resultant peripheral set 2 clock will have a period DIV1 + 1 times that of the system clock. If the DE1 is set to '0' (Divider 1 is disabled), any write access to the DIV1 field is ignored and the peripheral set 2 clock remains disabled.
DE2	**Divider 2 Enable** 0 Disable system clock divider 2 1 Enable system clock divider 2
DIV2	**Divider 2 Division Value** — The resultant peripheral set 3 clock will have a period DIV2 + 1 times that of the system clock. If the DE2 is set to '0' (Divider 2 is disabled), any write access to the DIV2 field is ignored and the peripheral set 3 clock remains disabled.

7.6 Functional Description

7.6.1 System Clock Generation

Figure 47 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 16 MHz int. RC osc. as the system clock and to ignore the system clock select.

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 normal operation, the system clock can be disabled by the MC_ME.

System Clock Dividers

The MC_CGM generates three derived clocks from the system clock.

Dividers Functional Description

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

- *Section System Clock Divider Configuration Registers (CGM_SC_DC0…2)*

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 DIVn field is ignored.

7.6.2 Output Clock Multiplexing

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

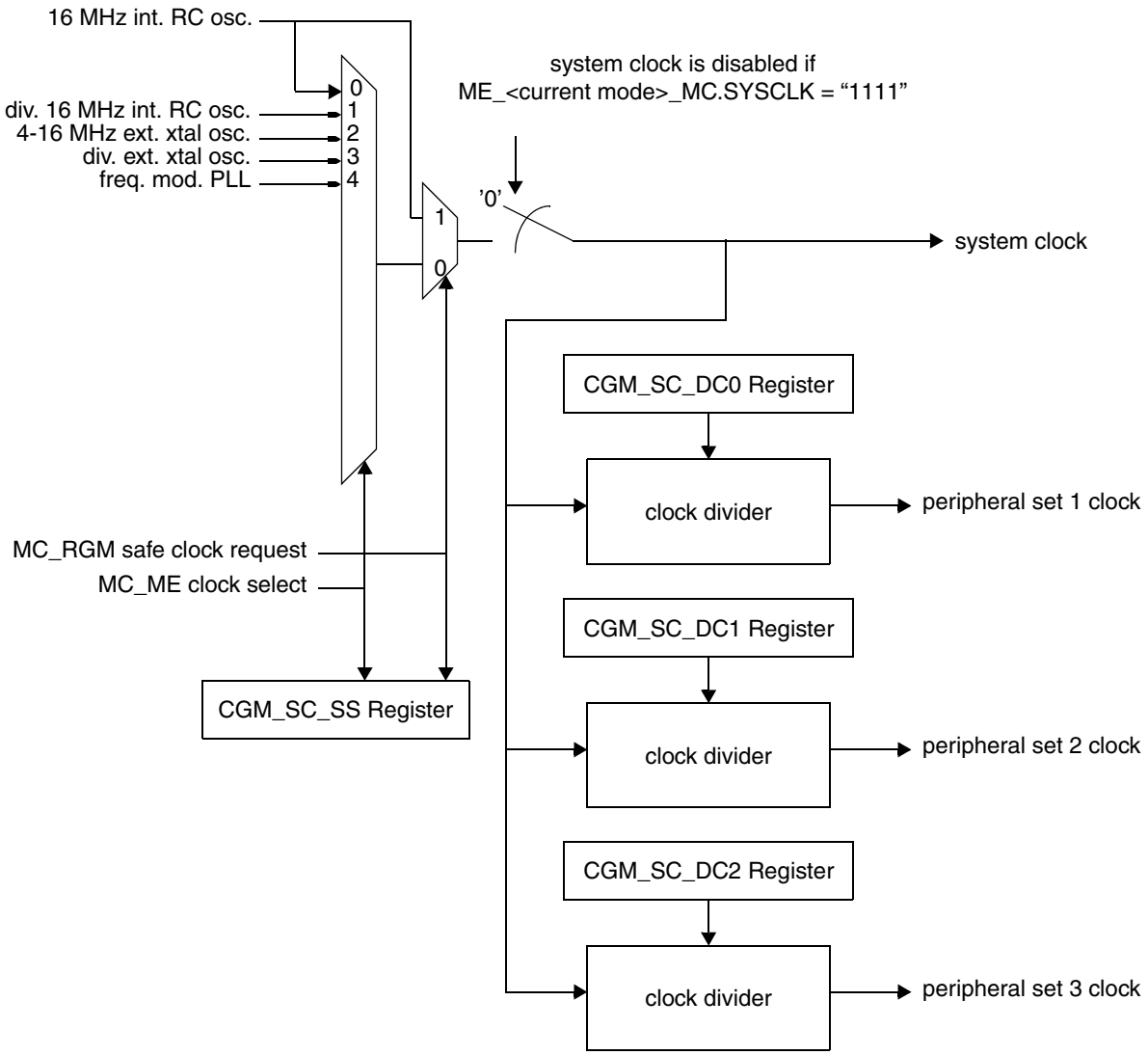


Figure 47. MC_CGM System Clock Generation Overview

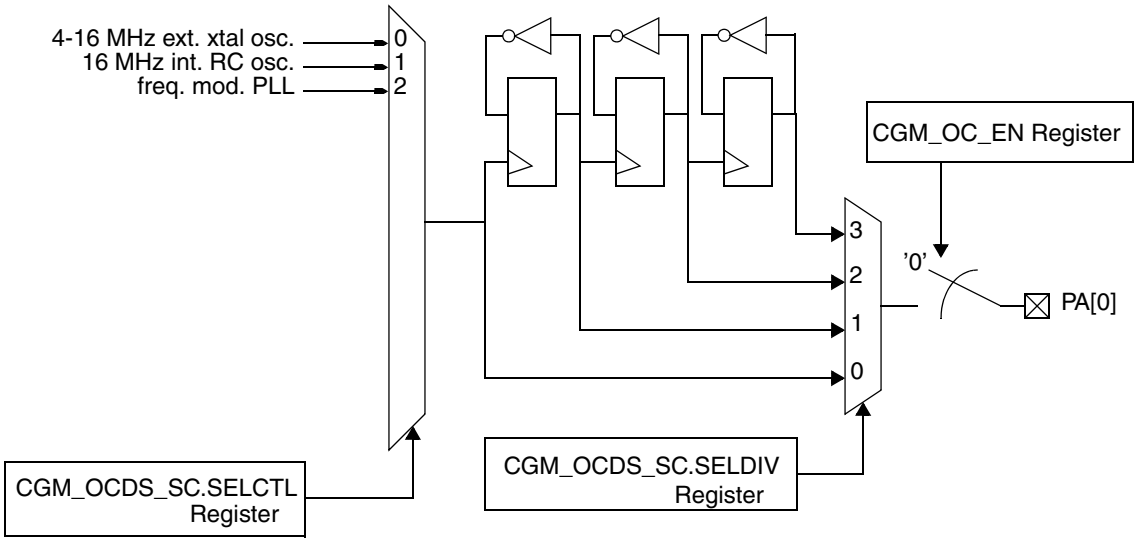


Figure 48. MC_CGM Output Clock Multiplexer and PA[0] Generation

7.6.3 Output Clock Division Selection

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

- PA[0] (see *Figure 48*). This signal is generated by utilizing 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)*) which contains the output clock enable/disable control bit.

8 Mode Entry Module (MC_ME)

8.1 Introduction

8.1.1 Overview

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

Figure 49 depicts the MC_ME block diagram.

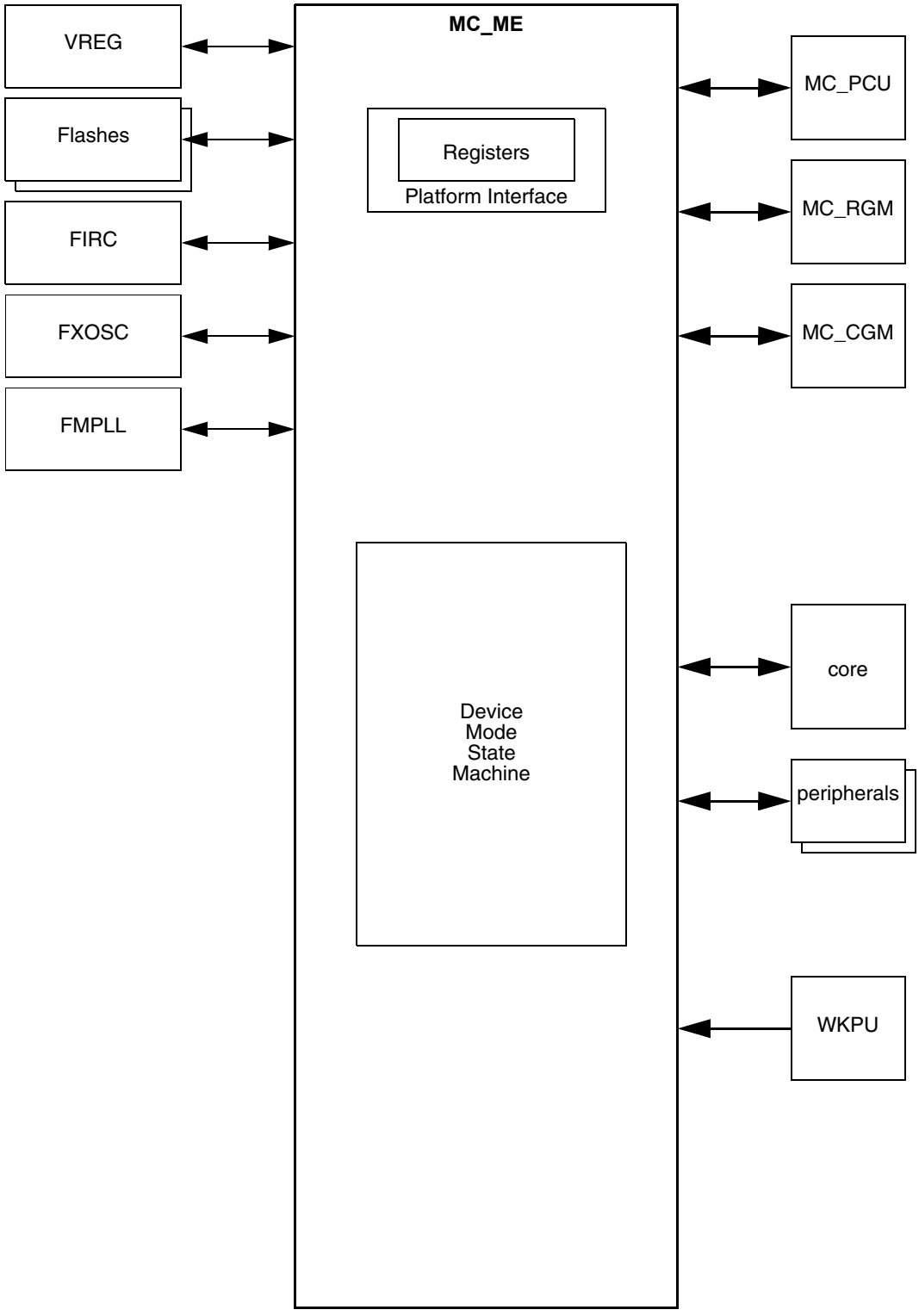


Figure 49. MC_MEBlock Diagram

8.1.2 Features

The MC_ME includes the following features:

- control of the available modes by the **ME_ME** register
- definition of various device mode configurations by the **ME_<mode>_MC** registers
- control of the actual device 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_PCTL0…143** registers
- capture of current peripheral clock gated/enabled status

8.1.3 Modes of Operation

The MC_ME is based on several device modes corresponding to different usage models of the device. 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, HALT**, **STOP**, and **STANDBY** 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 device software running modes.

Table 60 describes the MC_ME modes.

Table 60. 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 device. It manages hardware initialization of chip configuration, voltage regulators, oscillators, PLLs, 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**, wakeup request from **STANDBY**	system reset assertion, **RUN0…3**, **TEST**, **STANDBY** 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 60. 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 device self-test. It may enable the application to run its own self-test like flash checksum, memory BIST etc.	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**, interrupt event from **HALT**, interrupt or wakeup event from **STOP**	system reset assertion, **SAFE** via software or hardware failure, other **RUN0…3** modes, **HALT**, **STOP**, **STANDBY** via software
HALT	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 PLL, 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
STOP	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 oscillator 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
STANDBY	This is a reduced-leakage low-power mode during which power supply is cut off from most of the device. Wakeup from this mode takes a relatively long time, and content is lost or must be restored from backup.	software request from **RUN0…3**, **DRUN** modes	system reset assertion, **DRUN** on wakeup event

8.2 External Signal Description

The MC_ME has no connections to any external pins.

8.3 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

Table 61. MC_ME Register Description

Address	Name	Description	Size	Access	Location
				Supervisor	
0xC3FD_C000	ME_GS	Global Status	word	read	*on page 8-155*
0xC3FD_C004	ME_MCTL	Mode Control	word	read/write	*on page 8-157*

Table 61. MC_ME Register Description (continued)

Address	Name	Description	Size	Access	Location
				Supervisor	
0xC3FD_C008	ME_ME	Mode Enable	word	read/write	on page 8-158
0xC3FD_C00C	ME_IS	Interrupt Status	word	read/write	on page 8-160
0xC3FD_C010	ME_IM	Interrupt Mask	word	read/write	on page 8-161
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read/write	on page 8-162
0xC3FD_C018	ME_DMTS	Debug Mode Transtion Status	word	read	on page 8-163
0xC3FD_C020	ME_RESET_MC	**RESET** Mode Configuration	word	read	on page 8-165
0xC3FD_C024	ME_TEST_MC	**TEST** Mode Configuration	word	read/write	on page 8-166
0xC3FD_C028	ME_SAFE_MC	**SAFE** Mode Configuration	word	read/write	on page 8-166
0xC3FD_C02C	ME_DRUN_MC	**DRUN** Mode Configuration	word	read/write	on page 8-167
0xC3FD_C030	ME_RUN0_MC	**RUN0** Mode Configuration	word	read/write	on page 8-167
0xC3FD_C034	ME_RUN1_MC	**RUN1** Mode Configuration	word	read/write	on page 8-167
0xC3FD_C038	ME_RUN2_MC	**RUN2** Mode Configuration	word	read/write	on page 8-167
0xC3FD_C03C	ME_RUN3_MC	**RUN3** Mode Configuration	word	read/write	on page 8-167
0xC3FD_C040	ME_HALT_MC	**HALT** Mode Configuration	word	read/write	on page 8-168
0xC3FD_C048	ME_STOP_MC	**STOP** Mode Configuration	word	read/write	on page 8-168
0xC3FD_C054	ME_STANDBY_MC	**STANDBY** Mode Configuration	word	read/write	on page 8-169
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	on page 8-171
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	on page 8-171
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	on page 8-172
0xC3FD_C06C	ME_PS3	Peripheral Status 3	word	read	on page 8-172
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read/write	on page 8-173
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read/write	on page 8-173
...					
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read/write	on page 8-173
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read/write	on page 8-174
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read/write	on page 8-174
...					
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read/write	on page 8-174
0xC3FD_C0C4	ME_PCTL4	DSPI0 Control	byte	read/write	on page 8-175
0xC3FD_C0C5	ME_PCTL5	DSPI1 Control	byte	read/write	on page 8-175
0xC3FD_C0C6	ME_PCTL6	DSPI2 Control	byte	read/write	on page 8-175

Table 61. MC_ME Register Description (continued)

Address	Name	Description	Size	Access	Location
				Supervisor	
0xC3FD_C0D0	ME_PCTL16	FlexCAN0 Control	byte	read/write	*on page 8-175*
0xC3FD_C0D1	ME_PCTL17	FlexCAN1 Control	byte	read/write	*on page 8-175*
0xC3FD_C0D2	ME_PCTL18	FlexCAN2 Control	byte	read/write	*on page 8-175*
0xC3FD_C0D3	ME_PCTL19	FlexCAN3 Control	byte	read/write	*on page 8-175*
0xC3FD_C0D4	ME_PCTL20	FlexCAN4 Control	byte	read/write	*on page 8-175*
0xC3FD_C0D5	ME_PCTL21	FlexCAN5 Control	byte	read/write	*on page 8-175*
0xC3FD_C0E0	ME_PCTL32	ADC0 Control	byte	read/write	*on page 8-175*
0xC3FD_C0EC	ME_PCTL44	I2C0 Control	byte	read/write	*on page 8-175*
0xC3FD_C0F0	ME_PCTL48	LINFlex0 Control	byte	read/write	*on page 8-175*
0xC3FD_C0F1	ME_PCTL49	LINFlex1 Control	byte	read/write	*on page 8-175*
0xC3FD_C0F2	ME_PCTL50	LINFlex2 Control	byte	read/write	*on page 8-175*
0xC3FD_C0F3	ME_PCTL51	LINFlex3 Control	byte	read/write	*on page 8-175*
0xC3FD_C0F9	ME_PCTL57	CTU Control	byte	read/write	*on page 8-175*
0xC3FD_C0FC	ME_PCTL60	CANSampler Control	byte	read/write	*on page 8-175*
0xC3FD_C104	ME_PCTL68	SIUL Control	byte	read/write	*on page 8-175*
0xC3FD_C105	ME_PCTL69	WKPU Control	byte	read/write	*on page 8-175*
0xC3FD_C108	ME_PCTL72	eMIOS0 Control	byte	read/write	*on page 8-175*
0xC3FD_C109	ME_PCTL73	eMIOS1 Control	byte	read/write	*on page 8-175*
0xC3FD_C11B	ME_PCTL91	RTC_API Control	byte	read/write	*on page 8-175*
0xC3FD_C11C	ME_PCTL92	PIT_RTI Control	byte	read/write	*on page 8-175*
0xC3FD_C128	ME_PCTL104	CMU Control	byte	read/write	*on page 8-175*

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 62. MC_ME Memory Map

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
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	S_DC	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
		W																
		R										S_FMPLL	S_FXOSC	S_FIRC	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	STANDBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		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	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W													**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	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																
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**

Table 62. MC_ME Memory Map (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
0xC3FD_C018	ME_DMTS	R	0	0	0	0	0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
		W																
		R	0	FMPLL_SC	FXOSC_SC	FIRC_SC		SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	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										FMPLLON	FXOSCON	FIRCON	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	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R										FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C02C	ME_DRUN_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																

Table 62. MC_ME Memory Map (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
		R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	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	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C040	ME_HALT_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	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C044	reserved																	
0xC3FD_C048	ME_STOP_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	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C04C … 0xC3FD_C050	reserved																	
0xC3FD_C054	ME_STANDBY_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R										FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																

Table 62. MC_ME Memory Map (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
0xC3FD_C058 … 0xC3FD_C05C	reserved																	
0xC3FD_C060	ME_PS0	R											S_FlexCAN5	S_FlexCAN4	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
		W																
		R										S_DSPI2	S_DSPI1	S_DSPI0				
		W																
0xC3FD_C064	ME_PS1	R				S_CANSampler			S_CTU						S_LINFlex3	S_LINFlex2	S_LINFlex1	S_LINFlex0
		W																
		R				S_I2C0												S_ADC0
		W																
0xC3FD_C068	ME_PS2	R				S_PIT_RTI	S_RTC_API											
		W																
		R							S_eMIOS1	S_eMIOS0			S_WKPU	S_SIUL				
		W																
0xC3FD_C06C	ME_PS3	R																
		W																
		R								S_CMU								
		W																
0xC3FD_C070	reserved																	

Table 62. MC_ME Memory Map (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
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	STANDBY	0	0	STOP	0	HALT	0	0	0	0	0	0	0	0
		W																
0xC3FD_C0C0 … 0xC3FD_C14C	ME_PCTL0…143	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																	

8.3.1 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.

Global Status Register (ME_GS)

Figure 50. Global Status Register (ME_GS)

Address 0xC3FD_C000 Access: Supervisor read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	S_CURRENT_MODE				S_MTRANS	S_DC	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										S_FMPLL	S_FXOSC	S_FIRC	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 63. Global Status Register (ME_GS) Field Descriptions

Field	Description
S_CURRENT_MODE	**Current device mode status** 0000 **RESET** 0001 **TEST** 0010 **SAFE** 0011 **DRUN** 0100 **RUN0** 0101 **RUN1** 0110 **RUN2** 0111 **RUN3** 1000 **HALT** 1001 reserved 1010 **STOP** 1011 reserved 1100 reserved 1101 **STANDBY** 1110 reserved 1111 reserved
S_MTRANS	**Mode transition status** 0 Mode transition process is not active 1 Mode transition is ongoing
S_DC	**Device current consumption status** 0 Device consumption is low enough to allow powering down of main voltage regulator 1 Device consumption requires main voltage regulator to remain powered regardless of mode configuration

Table 63. 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 pads power sequence driver is disabled. The inputs are level unchanged. In **STOP** mode, only pad power sequence driver is disabled but the state of the output is kept. In **STANDBY** mode, the power sequence driver and all pads except those mapped on wakeup lines are not powered and therefore high impedance. Wakeup lines configuration remains unchanged
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 in low-power mode 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	**frequency modulated phase locked loop status** 0 frequency modulated phase locked loop is not stable 1 frequency modulated phase locked loop is providing a stable clock
S_FXOSC	**fast external crystal oscillator (4-16 MHz) status** 0 fast external crystal oscillator (4-16 MHz) is not stable 1 fast external crystal oscillator (4-16 MHz) is providing a stable clock
S_FIRC	**fast internal RC oscillator (16 MHz) status** 0 fast internal RC oscillator (16 MHz) is not stable 1 fast internal RC oscillator (16 MHz) is providing a stable clock
S_SYSCLK	**System clock switch status** — These bits specify the system clock currently used by the system. 0000 16 MHz int. RC osc. 0001 div. 16 MHz int. RC osc. 0010 4-16 MHz ext. xtal osc. 0011 div. ext. xtal osc. 0100 freq. mod. PLL 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 51. Mode Control Register (ME_MCTL)

Address 0xC3FD_C004 Access: Supervisor 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 64. Mode Control Register (ME_MCTL) Field Descriptions

Field	Description
TARGET_MODE	**Target device mode** — These bits provide the target device 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 **HALT** and **STOP** modes on hardware exit events, these are updated with the appropriate **RUN0…3** mode value. 0000 **RESET** 0001 **TEST** 0010 **SAFE** 0011 **DRUN** 0100 **RUN0** 0101 **RUN1** 0110 **RUN2** 0111 **RUN3** 1000 **HALT** 1001 reserved 1010 **STOP** 1011 reserved 1100 reserved 1101 **STANDBY** 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 52. Mode Enable Register (ME_ME)

Address 0xC3FD_C008 Access: Supervisor 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	STANDBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

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

Table 65. Mode Enable Register (ME_ME) Field Descriptions

Field	Description
STANDBY	**STANDBY mode enable** 0 **STANDBY** mode is disabled 1 **STANDBY** mode is enabled
STOP	**STOP mode enable** 0 **STOP** mode is disabled 1 **STOP** mode is enabled
HALT	**HALT mode enable** 0 **HALT** mode is disabled 1 **HALT** 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** 0 **RUN0** mode is disabled 1 **RUN0** mode is enabled
DRUN	**DRUN mode enable** 0 **DRUN** mode is disabled 1 **DRUN** mode is enabled
SAFE	**SAFE mode enable** 0 **SAFE** mode is disabled 1 **SAFE** mode is enabled
TEST	**TEST mode enable** 0 **TEST** mode is disabled 1 **TEST** mode is enabled
RESET	**RESET mode enable** 0 **RESET** mode is disabled 1 **RESET** mode is enabled

Interrupt Status Register (ME_IS)

Figure 53. Interrupt Status Register (ME_IS)

Address 0xC3FD_C00C Access: Supervisor 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	I_ICONF	I_IMODE	I_SAFE	I_MTC
W													**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 66. Interrupt Status Register (ME_IS) Field Descriptions

Field	Description
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 device 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 **HALT**, **STOP**, or **STANDBY**. 0 No mode transition complete interrupt occurred 1 Mode transition complete interrupt is pending

Interrupt Mask Register (ME_IM)

Figure 54. Interrupt Mask Register (ME_IM)

Address 0xC3FD_C010 Access: Supervisor 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	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 67. Interrupt Mask Register (ME_IM) Field Descriptions

Field	Description
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 55. Invalid Mode Transition Status Register (ME_IMTS)

Address 0xC3FD_C014 Access: Supervisor 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 each cause of invalid mode interrupt.

Table 68. 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 8.4.5 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 device 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 56. Debug Mode Transition Status Register (ME_DMTS)

Address 0xC3FD_C018 Access: Supervisor 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	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	FMPLL_SC	FXOSC_SC	FIRC_SC	0	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 69. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions

Field	Description
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

Table 69. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
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
FMPLL_SC	FMPLL State Change during mode transition indicator — This bit is set when the frequency modulated phase locked loop is requested to change its power up/down state. It is cleared when the frequency modulated phase locked loop has completed its state change. 0 No state change is taking place 1 A state change is taking place
FXOSC_SC	FXOSC State Change during mode transition indicator — This bit is set when the fast external crystal oscillator (4-16 MHz) is requested to change its power up/down state. It is cleared when the fast external crystal oscillator (4-16 MHz) has completed its state change. 0 No state change is taking place 1 A state change is taking place
FIRC_SC	FIRC State Change during mode transition indicator — This bit is set when the fast internal RC oscillator (16 MHz) is requested to change its power up/down state. It is cleared when the fast internal RC oscillator (16 MHz) has completed its state change. 0 No state change is taking place 1 A state change is taking place
SYSCLK_SW	System Clock Switching pending status — 0 No system clock source switching is pending 1 A 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 1 A 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 1 A state change is taking place
CDP_PRPH_0_143	Clock Disable Process Pending status for Peripherals 0…143 — 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 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_96_127	Clock Disable Process Pending status for Peripherals 96…127 — 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 1 Clock disabling is pending for at least one peripheral

Table 69. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64…95 — 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 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_32_63	Clock Disable Process Pending status for Peripherals 32…63 — 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 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_0_31	Clock Disable Process Pending status for Peripherals 0…31 — 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 1 Clock disabling is pending for at least one peripheral

RESET Mode Configuration Register (ME_RESET_MC)

Figure 57. Invalid Mode Transition Status Register (ME_IMTS)

Address 0xC3FD_C020 Access: Supervisor 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	FMPLLON	FXOSCON	FIRCON	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 70* for details.

TEST Mode Configuration Register (ME_TEST_MC)

Figure 58. TEST Mode Configuration Register (ME_TEST_MC)

Address 0xC3FD_C024 Access: Supervisor 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	FMPLLON	FXOSCON	FIRCON	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 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

SAFE Mode Configuration Register (ME_SAFE_MC)

Figure 59. SAFE Mode Configuration Register (ME_SAFE_MC)

Address 0xC3FD_C028 Access: Supervisor 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	FMPLLON	FXOSCON	FIRCON	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 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

DRUN Mode Configuration Register (ME_DRUN_MC)

Figure 60. DRUN Mode Configuration Register (ME_DRUN_MC)

Address 0xC3FD_C02C — Access: Supervisor 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	FMPLLON	FXOSCON	FIRCON	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 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

*Note: The values of **FXOSCON**, **CFLAON** and **DFLAON** are retained through **STANDBY** mode.*

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

Figure 61. RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC)

Address 0xC3FD_C030 - 0xC3FD_C03C — Access: Supervisor 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	FMPLLON	FXOSCON	FIRCON	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 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

HALT Mode Configuration Register (ME_HALT_MC)

Figure 62. HALT Mode Configuration Register (ME_HALT_MC)

Address 0xC3FD_C040 Access: Supervisor 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	0	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	FMPLLON	FXOSCON	FIRCON	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 **HALT** mode. Please refer to *Table 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

STOP Mode Configuration Register (ME_STOP_MC)

Figure 63. STOP Mode Configuration Register (ME_STOP_MC)

Address 0xC3FD_C048 Access: Supervisor 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	FMPLLON	FXOSCON	FIRCON	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 **STOP** mode. Please refer to *Table 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

STANDBY Mode Configuration Register (ME_STANDBY_MC)

Figure 64. STANDBY Mode Configuration Register (ME_STANDBY_MC)

Address 0xC3FD_C054 Access: Supervisor 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	0	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	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

This register configures system behavior during **STANDBY** mode. Please refer to *Table 70* for details.

Note: Byte and half-word write accesses are not allowed to this register.

Table 70. 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 **STOP** mode, only pad power sequence driver is disabled but the state of the output is kept. In **STANDBY** mode, power sequence driver and all pads except those mapped on wakeup lines are not powered and therefore high impedance. Wakeup line configuration remains unchanged.
MVRON	**Main voltage regulator control** — This bit specifies whether main voltage regulator is switched off or not while entering this mode. 0 Main voltage regulator is switched off 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 Data flash is in low-power mode 11 Data flash is in normal mode *Note: If the flash memory is to be powered down in any mode, then your software must ensure that reset sources are configured as long resets in the **RGM_FESS** register (see Section , Functional Event Short Sequence Register (RGM_FESS)).*

Table 70. Mode Configuration Registers (ME_<*mode*>_MC) Field Descriptions (continued)

Field	Description
CFLAON	**Code flash power-down control** — This bit specifies the operating mode of the program 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
FMPLLON	**frequency modulated phase locked loop control** 0 frequency modulated phase locked loop is switched off 1 frequency modulated phase locked loop is switched on
FXOSCON	**fast external crystal oscillator (4-16 MHz) control** 0 fast external crystal oscillator (4-16 MHz) is switched off 1 fast external crystal oscillator (4-16 MHz) is switched on
FIRCON	**fast internal RC oscillator (16 MHz) control** 0 fast internal RC oscillator (16 MHz) is switched off 1 fast internal RC oscillator (16 MHz) is switched on
SYSCLK	**System clock switch control** — These bits specify the system clock to be used by the system. 0000 16 MHz int. RC osc. 0001 div. 16 MHz int. RC osc. 0010 4-16 MHz ext. xtal osc. 0011 div. ext. xtal osc. 0100 freq. mod. PLL 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

Peripheral Status Register 0 (ME_PS0)

Figure 65. Peripheral Status Register 0 (ME_PS0)

Address 0xC3FD_C060 Access: Supervisor 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	S_FlexCAN5	S_FlexCAN4	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
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	S_DSPI2	S_DSPI1	S_DSPI0	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 71* for details.

Peripheral Status Register 1 (ME_PS1)

Figure 66. Peripheral Status Register 1 (ME_PS1)

Address 0xC3FD_C064 Access: Supervisor read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_CANSampler	0	0	S_CTU	0	0	0	0	0	S_LINFlex3	S_LINFlex2	S_LINFlex1	S_LINFlex0
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	S_I2C0	0	0	0	0	0	0	0	0	0	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 71* for details.

Peripheral Status Register 2 (ME_PS2)

Figure 67. Peripheral Status Register 2 (ME_PS2)

Address 0xC3FD_C068 Access: Supervisor read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_PIT_RTI	S_RTC_API	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	S_eMIOS1	S_eMIOS0	0	0	S_WKPU	S_SIUL	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 71* for details.

Peripheral Status Register 3 (ME_PS3)

Figure 68. Peripheral Status Register 3 (ME_PS3)

Address 0xC3FD_C06C Access: Supervisor 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	S_CMU	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 71* for details.

Table 71. Peripheral Status Registers 0…4 (ME_PS0…4) Field Descriptions

Field	Description
S_<*periph*>	**Peripheral status** — These bits specify the current status of the peripherals in the system. If no peripheral is mapped on a particular position, the corresponding bit is always read as '0'. 0 Peripheral is frozen 1 Peripheral is active

Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Figure 69. Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Address 0xC3FD_C080 - 0xC3FD_C09C Access: Supervisor 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 72. 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

Table 72. Run Peripheral Configuration Registers (ME_RUN_PC0…7) Field Descriptions

Field	Description
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 70. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

Address 0xC3FD_C0A0 - 0xC3FD_C0BC Access: Supervisor 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
W			STANDBY			STOP		HALT								
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 73. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) Field Descriptions

Field	Description
STANDBY	**Peripheral control during STANDBY** 0 Peripheral is frozen with clock gated 1 Peripheral is active
STOP	**Peripheral control during STOP** 0 Peripheral is frozen with clock gated 1 Peripheral is active
HALT	**Peripheral control during HALT** 0 Peripheral is frozen with clock gated 1 Peripheral is active

Peripheral Control Registers (ME_PCTL0…143)

Figure 71. Peripheral Control Registers (ME_PCTL0…143)

Address 0xC3FD_C0C0 - 0xC3FD_C14F Access: Supervisor 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.

Table 74. Peripheral Control Registers (ME_PCTL0…143) 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 device mode. 1 Peripheral is frozen if not already frozen in device 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.*
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

Table 75. Peripheral control registers by peripheral

Peripheral	ME_PCTLn
ADC_0	32
CAN sampler	60
CMU	104
CTU	57

Table 75. Peripheral control registers by peripheral (continued)

Peripheral	ME_PCTLn
DMA_MUX	23
DSPI_0	4
DSPI_1	5
DSPI_2	6
DSPI_3	7
eMIOS_0	72
eMIOS_1	73
FlexCAN_0	16
FlexCAN_1	17
FlexCAN_2	18
FlexCAN_3	10
FlexCAN_4	20
FlexCAN_5	21
I2C	44
LINFlex_0	48
LINFlex_1	49
LINFlex_2	50
LINFlex_3	51
PIT	92
RTC/API	91
SIUL	68
WKPU	69

8.4 Functional Description

8.4.1 Mode Transition Request

The transition from one mode to another mode is normally handled by software by accessing the mode control **ME_MCTL** register. But in case of special events, mode transition can be automatically managed by hardware. In order to switch from one mode to another, the application should access **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 8.4.5 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 **HALT**, **STOP**, or **STANDBY** modes.*

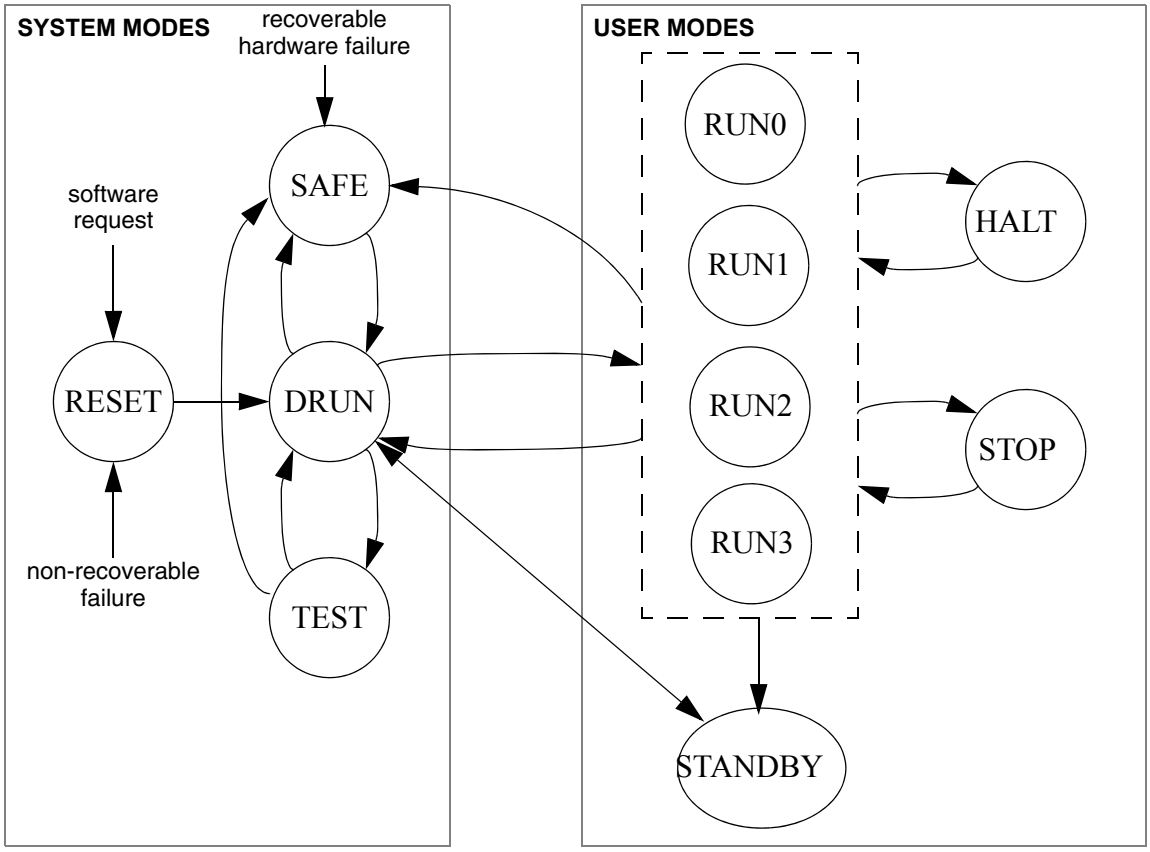


Figure 72. MC_ME Mode Diagram

8.4.2 Modes Details

RESET Mode

The device 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”
- 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 16 MHz int. RC osc. is selected as the system clock. All power domains are made active in this mode.

DRUN Mode

The device 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”
- from the **STANDBY** mode after an external wakeup event or internal wakeup alarm (e.g. RTC/API event)

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 16 MHz int. RC osc. as the system clock.

This mode is intended to be used by software

- to initialize all registers as per the system needs
- to execute small routines in a ‘ping-pong’ with the **STANDBY** mode

When this mode is entered from **STANDBY** after a wakeup event, the **ME_DRUN_MC** register content is restored to its pre-**STANDBY** values, and the mode starts in that configuration.

All power domains are active when this mode is entered due to a system reset sequence initiated by a destructive reset event. In other cases of entry, such as the exit from **STANDBY** after a wakeup event, a functional reset event like an external reset or a software request from **RUN0…3**, **SAFE**, or **TEST** mode, active power domains are determined by the power configuration register **PCU_PCONF2** of the MC_PCU. All power domains except power domains #0 and #1 are configurable in this mode (see the MC_PCU chapter for details).

Note: As flashes can be configured in low-power or power-down state in this mode, software must ensure that the code executes from SRAM before changing to this mode.

SAFE Mode

The device 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)

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 16 MHz int. RC osc. is selected as the system clock. All power domains are made active in this mode.

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. In this case, the new mode request is not interpreted as an invalid request.

*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 device’s final mode after mode transition will be the parent mode. However, this is not recommended software behavior. It is recommended for software to wait until the **S_MTRANS** bit is cleared after requesting a change to **SAFE** before requesting another mode change.*

As long as a **SAFE** event is active, the system remains in the **SAFE** mode and no write access is allowed to the **ME_MCTL** register.

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 device via the **DRUN** mode, or
  - completely reset the device 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. In this case, the pads’ power sequence driver cell is also disabled. The input levels remain unchanged.

TEST Mode

The device 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 device reset.

This mode is intended to be used by software

- to execute on-chip test routines

All power domains except power domains #0 and #1 are configurable in this mode. Active power domains are determined by the power configuration register **PCU_PCONF2** of the MC_PCU.

Note: As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from SRAM before it changes to this mode.

RUN0…3 Modes

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

- from the **DRUN** another **RUN0…3** mode when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “0100…0111”
- from the **HALT** mode by an interrupt event
- from the **STOP** mode by an interrupt or wakeup event

As soon as any of the above events occur, a **RUN0…3** mode transition request is generated. The mode configuration information for these modes is provided by **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

All power domains except power domains #0 and #1 are configurable in these modes in order to reduce leakage consumption. Active power domains are determined by the power configuration register **PCU_PCONF2** of the MC_PCU.

Note: As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from SRAM before it changes to this mode.

HALT Mode

The device 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 occur, a **HALT** mode transition request is generated. The mode configuration information for this mode is provided by **ME_HALT_MC** register. This mode is quite configurable, and the **ME_HALT_MC** register should be programmed according to the system needs. The main voltage regulator and the flashes can be put in power-down mode as needed. If there is a **HALT** mode request while an interrupt request is active, the device mode does not change, 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)

All power domains except power domains #0 and #1 are configurable in this mode in order to reduce leakage consumption. Active power domains are determined by the power configuration register **PCU_PCONF2** of the MC_PCU.

STOP Mode

The device 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 occur, a **STOP** mode transition request is generated. The mode configuration information for this mode is provided by the **ME_STOP_MC** register. This mode is fully configurable, and the **ME_STOP_MC** register should be programmed according to the system needs. The FMPLL is switched off in this mode. The main voltage regulator and the flashes can be put in power-down mode as needed. If there is a **STOP** mode request while any interrupt or wakeup event is active, the device mode does not change, 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)

If the pads’ power sequence driver cell needs to be disabled while entering this mode, the **PDO** bit of the **ME_STOP_MC** register should be set. The state of the outputs is kept.

This mode can be used to stop all clock sources, thus preserving the device status. When exiting the **STOP** mode, the fast internal RC oscillator (16 MHz) clock is selected as the system clock until the target clock is available.

All power domains except power domains #0 and #1 are configurable in this mode in order to reduce leakage consumption. Active power domains are determined by the power configuration register **PCU_PCONF2** of the MC_PCU.

STANDBY Mode

The device enters this mode on the following events:

- from the **DRUN** or one of the **RUN0…3** modes when the **TARGET_MODE** bit field of the **ME_MCTL** register is written with “1101”.

As soon as any of the above events occur, a **STANDBY** mode transition request is generated. The mode configuration information for this mode is provided by the **ME_STANDBY_MC** register. In this mode, the power supply is turned off for most of the device. The only parts of the device that are still powered during this mode are pads mapped on wakeup lines and power domain #0 which contains the MC_RGM, MC_PCU, WKPU, 8K RAM, RTC_API, CANSampler, SIRC, FIRC, SXOSC, and device and user option bits. The FIRC can be optionally switched off. This is the lowest power consumption mode possible on the device.

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

- the core, the flashes, and almost all peripherals and memories powered down

and to be used by software

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

The exit sequence of this mode is similar to the reset sequence. However, in addition to booting from the default location, the device can also be configured to boot from the backup SRAM (see the **RGM_STDBY** register description in the MC_RGM chapter for details). In the case of booting from backup SRAM, it is also possible to keep the flashes disabled by writing “01” to the **CFLAON** and **DFLAON** fileds in the **ME_DRUN_MC** register prior to **STANDBY** entry.

If there is a **STANDBY** mode request while any wakeup event is active, the device mode does not change.

All power domains except power domain #0 are configurable in this mode in order to reduce leakage consumption. Active power domains are determined by the power configuration register **PCU_PCONF2** of the MC_PCU.

8.4.3 Mode Transition Process

The process of mode transition follows the following steps in a pre-defined manner depending on the current device 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 valid 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 8.4.5 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 which can be generated 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*, 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 *Table 76*. A ‘√’ indicates that a given resource is configurable for a given mode.

Table 76. MC_ME Resource Control Overview

Resource	Mode							
	RESET	TEST	SAFE	DRUN	RUN0…3	HALT	STOP	STANDBY
FIRC	on	√ on	on	on	on	√ on	√ on	√ on
FXOSC	off	√ off	off	√ off	√ off	√ off	√ off	off
FMPLL	off	√ off	off	√ off	√ off	√ off	off	off
CFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down	power-down
DFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down	power-down
MVREG	on	on	on	on	on	√ on	√ on	off
PDO	off	√ off	√ on	off	off	off	√ off	on

Peripheral Clocks Disable

On completion of the *Target Mode Request*, 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_PCTL0…143**

Caution: The MC_ME does not automatically request 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. Therefore, it is software's responsibility 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 8.4.6 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 device enters the **SAFE** mode.

Processor Low-Power Mode Entry

If, on completion of the *Peripheral Clocks Disable*, the mode transition is to the **HALT** 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*, the mode transition is to the **STOP** or **STANDBY** 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*, the mode transition is to the **HALT**, **STOP**, or **STANDBY** 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 memories are unaffected for all transitions between software running modes including **DRUN**, **RUN0…3**, and **SAFE**.

Caution: Clocks to the whole device including the processor and system memories can be disabled in **TEST** mode.

Clock Sources Switch-On

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

- the fast internal RC oscillator (16 MHz)
- the fast external crystal oscillator (4-16 MHz)

Note: The frequency modulated phase locked loop, which needs the main voltage regulator to be stable, is not controlled by this step.

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 system 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.

Main Voltage Regulator Switch-On

On completion of the *Target Mode Request*, if the main voltage regulator needs to be switched on from its off state based on the **MVRON** bit of the **ME_<current mode>_MC** and **ME_<target mode>_MC** registers, the MC_ME requests the MC_PCU to power-up the regulator and waits for the output voltage stable status in order to update the **S_MVR** bit of the **ME_GS** register.

This step is required only during the exit of the low-power modes **HALT** and **STOP**. In this step, the fast internal RC oscillator (16 MHz) is switched on regardless of the target mode

configuration, as the main voltage regulator requires the 16 MHz int. RC osc. during power-up in order to generate the voltage status.

During the **STANDBY** exit sequence, the MC_PCU alone manages the power-up of the main voltage regulator, and the MC_ME is kept in **RESET** or shut off (depending on the power domain #1 status).

Flash Modules Switch-On

On completion of the *Main Voltage Regulator Switch-On*, if a flash module 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 flash modules are available for access, the **S_CFLA** and **S_DFLA** bit fields of the **ME_GS** register are updated to “11” by hardware.

If the main regulator is also off in device low-power modes, then during the exit sequence, the flash is kept in its low-power state and is switched on only when the *Main Voltage Regulator Switch-On* process has completed.

Caution: It is illegal to switch the flashes 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.

FMPLL Switch-On

On completion of the *Clock Sources Switch-On* and *Main Voltage Regulator Switch-On*, if the FMPLL is to be switched on from the off state based on the **FMPLLON** bit of the **ME_<current mode>_MC** and **ME_<target mode>_MC** registers, the MC_ME requests the FMPLL digital interface to start the phase locking process and waits for the FMPLL to enter into the locked state. When the FMPLL enters the locked state and starts providing a stable output clock, the **S_FMPLL** bit of **ME_GS** register is set.

Power Domain #2 Switch-On

On completion of the *Main Voltage Regulator Switch-On*, the MC_ME indicates a mode change to the MC_PCU. The MC_PCU then determines whether a power-up sequence is required for power domain #2. Only after the MC_PCU has executed all required power-ups does the MC_ME complete the mode transition.

Pad Outputs-On

On completion of the *Main Voltage Regulator Switch-On*, 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 device modes, the peripheral configuration registers **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and the peripheral control registers **ME_PCTL0…143**, the MC_ME enables the clocks for selected modules as required. This step is executed only after the *Main Voltage Regulator Switch-On* process is completed.

Also if a mode change translates to a power up of one or more power domains, the MC_PCU indicates the MC_ME after completing the power-up sequence upon which the

MC_ME may assert the peripheral clock enables of the peripherals residing in those power domains.

Processor and Memory Clock Enable

If the mode transition is from any of the low-power modes **HALT** or **STOP** to **RUN0…3**, the clocks to the processor and system memories 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 **HALT**, **STOP**, or **STANDBY** 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 16 MHz int. RC osc. is effective only when the **S_FIRC** bit of the **ME_GS** register is set by hardware (i.e. the fast internal RC oscillator (16 MHz) has stabilized).
- The target clock configuration for the div. 16 MHz int. RC osc. is effective only when the **S_FIRC** bit of the **ME_GS** register is set by hardware (i.e. the fast internal RC oscillator (16 MHz) has stabilized).
- The target clock configuration for the 4-16 MHz ext. xtal osc. is effective only when the **S_FXOSC** bit of the **ME_GS** register is set by hardware (i.e the fast external crystal oscillator (4-16 MHz) has stabilized).
- The target clock configuration for the div. ext. xtal osc. is effective only when the **S_FXOSC** bit of the **ME_GS** register is set by hardware (i.e the fast external crystal oscillator (4-16 MHz) has stabilized).
- The target clock configuration for the freq. mod. PLL is effective only when the **S_FMPLL** bit of the **ME_GS** register is set by hardware (i.e. the frequency modulated phase locked loop has stabilized).
- If the clock is to be disabled, the **SYSCLK** bit field should be programmed with “1111”. This is possible only in the **STOP** and **TEST** modes. In the **STANDBY** mode, the clock configuration is fixed, and the system clock is automatically forced to ‘0’.

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 needs to be switched on
- the *FMPLL Switch-On* process has completed if the target system clock is the *freq. mod. PLL*
- the *Peripheral Clocks Disable* process is 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 77*. A ‘√’ indicates that a given clock source is selectable for a given mode.

Table 77. MC_ME System Clock Selection Overview

System Clock Source	Mode							
	RESET	TEST	SAFE	DRUN	RUN0…3	HALT	STOP	STANDBY
16 MHz int. RC osc.	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	
div. 16 MHz int. RC osc.		√		√	√	√	√	
4-16 MHz ext. xtal osc.		√		√	√	√	√	
div. ext. xtal osc.		√		√	√	√	√	
freq. mod. PLL		√		√	√	√		
system clock is disabled		√(1)					√	√ (default)

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

Power Domain #2 Switch-Off

Based on the device mode and the MC_PCU's power configuration register **PCU_PCONF2**, the power domain #2 is controlled by the MC_PCU.

If a mode change translates to a power-down of the power domain, then the MC_PCU starts the power-down sequence. The MC_PCU acknowledges the completion of the power-down sequence with respect to the new mode, and the MC_ME uses this information to update the mode transition status. This step is executed only after the *Peripheral Clocks Disable* process has completed.

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**
- I/O pads power sequence driver is switched off if the target mode is one of **SAFE**, **TEST**, or **STOP** modes

In STANDBY mode, the power sequence driver and all pads except the external reset and those mapped on wakeup lines are not powered and therefore high impedance. The wakeup line configuration remains unchanged.

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

FMPLL Switch-Off

Based on the **FMPLLON** bit of the **ME_<current mode>_MC** and **ME_<target mode>_MC** registers, if FMPLL is to be switched off, the MC_ME requests the FMPLL to power down

and updates its availability status bit **S_FMPLL** of the **ME_GS** register to '0'. This step is executed only after the *System Clock Switching* process is completed.

Clock Sources Switch-Off

Based on the device mode and the **<clock source>ON** bits of the **ME_<mode>_MC** registers, if a given clock source is to be 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'.

This step is executed only after

- *System Clock Switching* process is completed in order not to lose the current system clock during mode transition.
- *FMPLL Switch-Off* as the input reference clock of the FMPLL can be among these clock sources. This is needed to prevent an unwanted lock transition when the FMPLL is switched on.

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 flash modules is to be put in a low-power state, the MC_ME requests the flash to enter the corresponding low-power state and waits for the deassertion of flash ready status signal. The exact low-power mode status of the flash modules is updated in the **S_CFLA** and **S_DFLA** bit fields of the **ME_GS** register. This step is executed only when *Processor and System Memory Clock Disable* process is completed.

Main Voltage Regulator Switch-Off

Based on the **MVRON** bit of the **ME_<current mode>_MC** and **ME_<target mode>_MC** registers, if the main voltage regulator is to be switched off, the MC_ME requests it to power down and clears the availability status bit **S_MVR** of the **ME_GS** register.

This step is required only during the entry of low-power modes like **HALT** and **STOP**. This step is executed only after completing the following processes:

- *FMPLL Switch-Off*
- *Flash Switch-Off*
- *Power Domain #2 Switch-Off*
- *Power Domain #2 Switch-On*
- the device consumption is less than the pre-defined threshold value (i.e. the S_DC bit of the **ME_GS** register is '0').

If the target mode is **STANDBY**, the main voltage regulator is not switched off by the MC_ME and the **STANDBY** request is asserted after the above processes have completed upon which the MC_PCU takes control of the main regulator. As the MC_PCU needs the 16 MHz int. RC osc., the fast internal RC oscillator (16 MHz) remains active until all the **STANDBY** steps are executed by the MC_PCU after which it may be switched off depending on the **FIRCON** bit of the **ME_STANDBY_MC** register.

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

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.

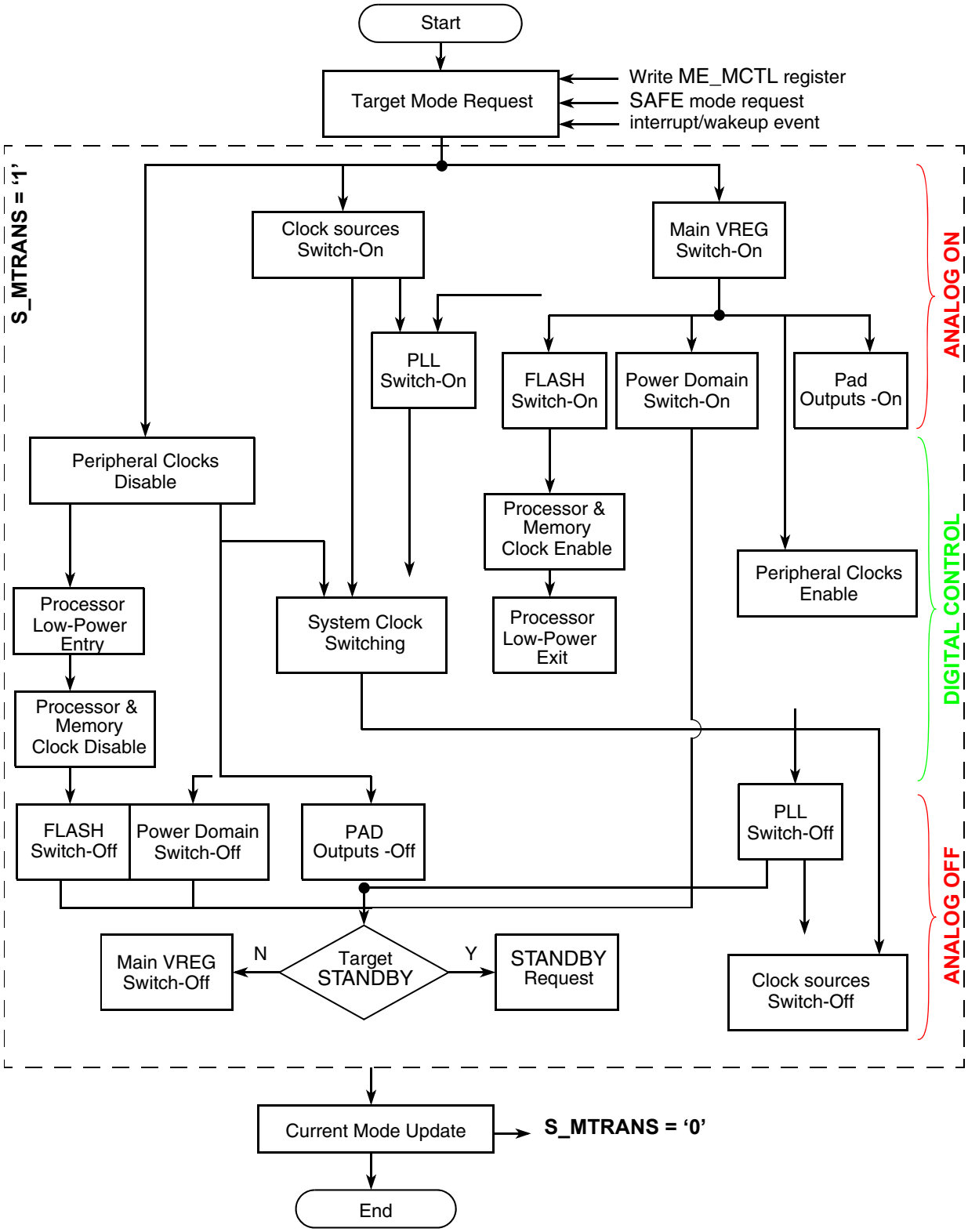


Figure 73. MC_ME Transition Diagram

8.4.4 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.

- FIRC must be on if the system clock is one of the following:
  - 16 MHz int. RC osc.
  - div. 16 MHz int. RC osc.
- FXOSC must be on if the system clock is one of the following:
  - 4-16 MHz ext. xtal osc.
  - div. ext. xtal osc.

Note: Software must ensure to switch on the clock source that provides the input reference clock to the FMPLL. There is no automatic protection mechanism to check this in the MC_ME.

- FMPLL must be on if the system clock is the freq. mod. PLL.
- Configuration “00” for the **CFLAON** and **DFLAON** bit fields are reserved.
- MVREG must be on if any of the following is active:
  - FMPLL
  - CFLASH
  - DFLASH
- System clock configurations marked as ‘reserved’ may not be selected.
- Configuration “1111” for the **SYSCLK** bit field is allowed only for the **STOP** and **TEST** modes, and only in this case may all system clock sources be turned off.

Caution: If the system clock is stopped during TEST mode, the device can exit only via a system reset.

8.4.5 Mode Transition Interrupts

The following are the three interrupts related to mode transition implemented in the MC_ME.

Invalid Mode Configuration Interrupt

Whenever a write operation is attempted to the **ME_<mode>_MC** registers violating the protection rules mentioned in the *Section 8.4.4 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 **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 device 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 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 **HALT** and **STOP** 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 device 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 device 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** is **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 completes a mode transition fully (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 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 **HALT** and **STOP** in order to avoid the same event requesting the exit of these low-power modes.

8.4.6 Peripheral Clock Gating

During all device 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 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_PCTL0…143** registers.

The low-power peripheral configuration registers **ME_LP_PC0…7** are chosen only during the low-power modes **HALT**, **STOP**, and **STANDBY**. 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_PCTL0…143** registers.

Any modifications to the **ME_RUN_PC0…7**, **ME_LP_PC0…7**, and **ME_PCTL0…143** 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_PCTL0…143** 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_PCTL0…143** registers during a debug session will take affect immediately without requiring any new mode request.

8.4.7 Application Example

Figure 74 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

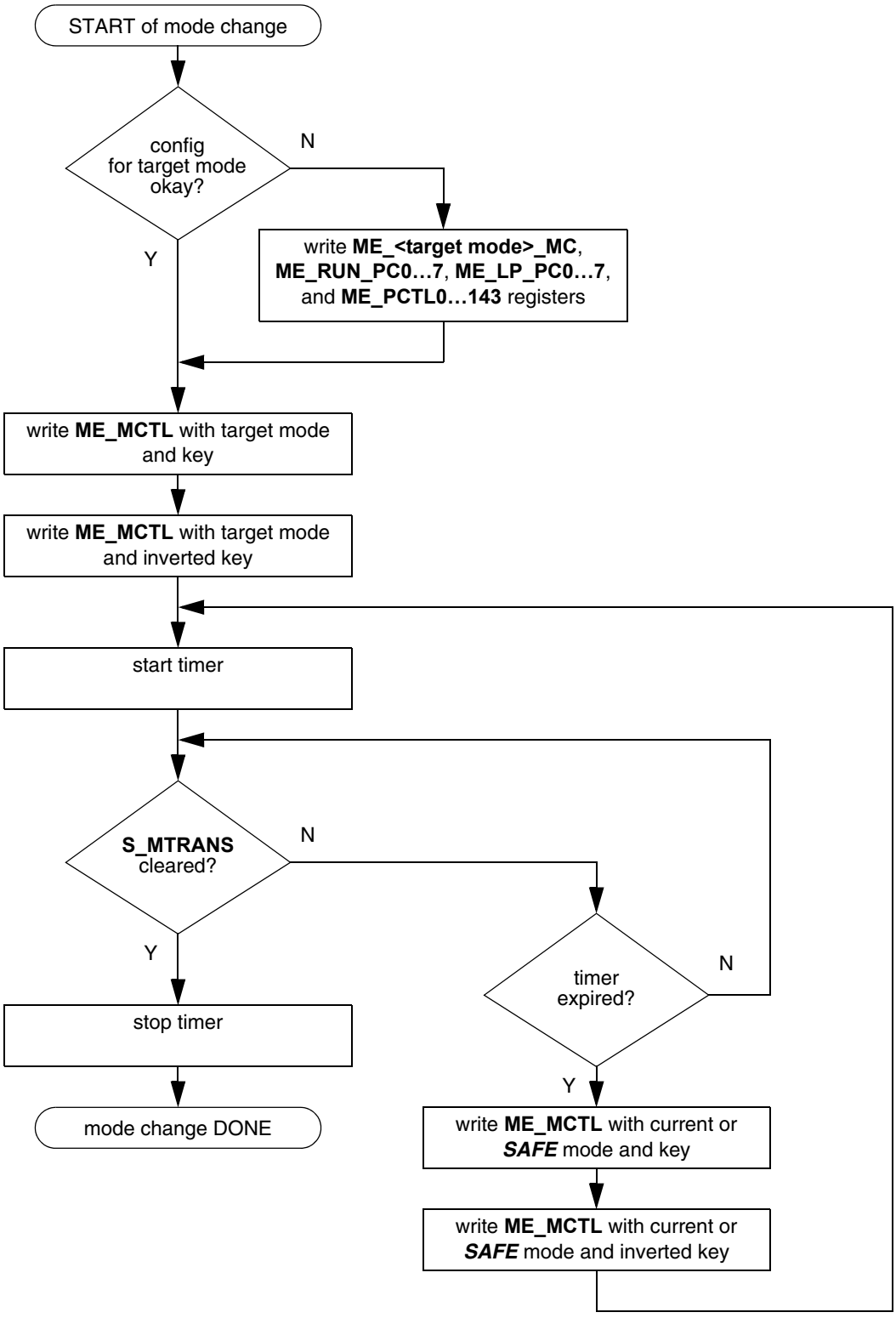


Figure 74. MC_ME Application Example Flow Diagram

9 Reset Generation Module (MC_RGM)

9.1 Introduction

9.1.1 Overview

The reset generation module (MC_RGM) centralizes the different reset sources and manages the reset sequence of the device. It provides a register interface and the reset sequencer. The different registers are available to monitor and control the device 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 control the reset signals generated in the system.

Figure 75 depicts the MC_RGM block diagram.

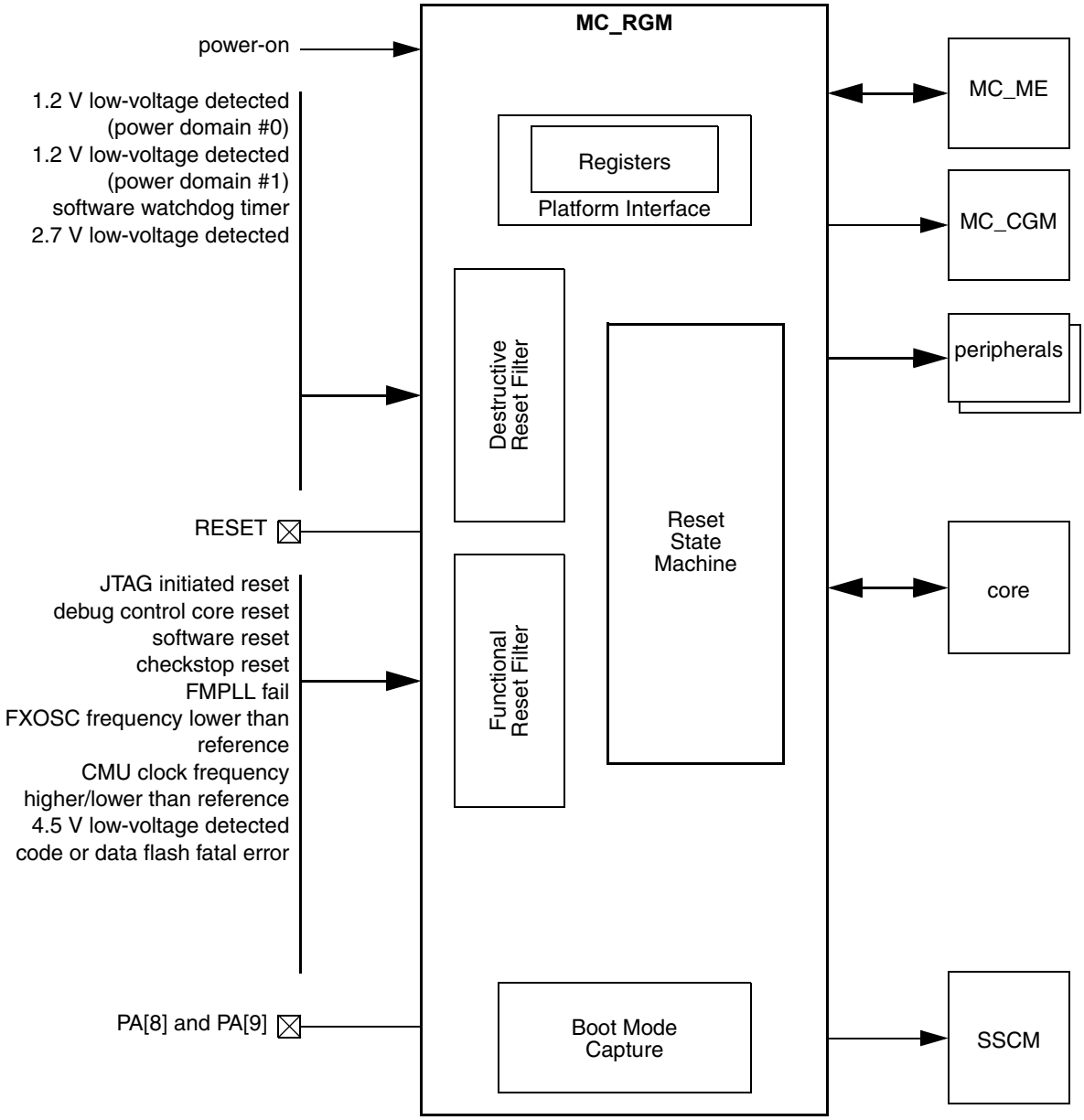


Figure 75. MC_RGM block diagram

9.1.2 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 (for further mode details, please see the MC_ME chapter)
- short reset sequence configuration
- bidirectional reset behavior configuration
- selection of alternate boot via the backup SRAM on STANDBY mode exit (for further mode details, please see the MC_ME chapter)
- boot mode capture on RESET deassertion

9.1.3 Modes of operation

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 device starting from PHASE0. This resets the full device ensuring a safe start-up state for both digital and analog modules. 'Destructive' resets are
  - power-on reset
  - 1.2 V low-voltage detected (power domain #0)
  - 1.2 V low-voltage detected (power domain #1)
  - software watchdog timer
  - 2.7 V low-voltage detected
- 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 device starting from PHASE1. In this case, most digital modules are reset normally, while analog modules or specific digital modules' (e.g. debug modules, flash modules) state is preserved. 'Functional' resets are
  - external reset
  - JTAG initiated reset
  - debug control core reset
  - software reset
  - checkstop reset
  - FMPLL fail
  - FXOSC frequency lower than reference
  - CMU clock frequency higher/lower than reference
  - 4.5 V low-voltage detected
  - code or data flash fatal error

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 device 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

device 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 Destructive Event Reset Disable Register (RGM_DERD)* and *Section Destructive Event Alternate Request Register (RGM_DEAR)* for 'destructive' resets and *Section Functional Event Reset Disable Register (RGM_FERD)* and *Section Functional Event Alternate Request Register (RGM_FEAR)* for 'functional' resets).

9.2 External signal description

The MC_RGM interfaces to the bidirectional reset pin RESET and the boot mode pins PA[8] and PA[9].

9.3 Memory map and register definition

Table 78. MC_RGM register description

Address	Name	Description	Size	Access	Location
				Supervisor	
0xC3FE_4000	RGM_FES	Functional Event Status	half-word	read/write[1]	*on page 9-201*
0xC3FE_4002	RGM_DES	Destructive Event Status	half-word	read/write(1)	*on page 9-202*
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	half-word	read/write(2)	*on page 9-204*
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	half-word	read	*on page 9-205*
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	half-word	read/write	*on page 9-206*
0xC3FE_4012	RGM_DEAR	Destructive Event Alternate Request	half-word	read	*on page 9-208*
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	half-word	read/write	*on page 9-209*
0xC3FE_401A	RGM_STDBY	STANDBY Reset Sequence	half-word	read/write	*on page 9-210*
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	half-word	read/write	*on page 9-211*

1. individual bits cleared on writing '1'
2. write once: '0' = disable, '1' = enable.

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 79. MC_RGM Memory Map

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_4000	RGM_FES / RGM_DES	R	F_EXR	0	0	0	0	0	0	F_FLASH	F_LVD45	F_CMU_FHL	F_CMU_OLR	F_FMPLL	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
		W	w1c							w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	0	0	0	0	0	0	0	0	0	0	0	F_LVD27	F_SWT	F_LVD12_PD1	F_LVD12_PD0
		W	w1c												w1c	w1c	w1c	w1c
0xC3FE_4004	RGM_FERD / RGM_DERD	R	D_EXR	0	0	0	0	0	0	D_FLASH	D_LVD45	D_CMU_FHL	D_CMU_OLR	D_FMPLL	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	D_LVD27	D_SWT	D_LVD12_PD1	D_LVD12_PD0
		W																
0xC3FE_4008 … 0xC3FE_400C	reserved																	
0xC3FE_4010	RGM_FEAR / RGM_DEAR	R	AR_EXR	0	0	0	0	0	0	AR_FLASH	AR_LVD45	AR_CMU_FHL	AR_CMU_OLR	AR_FMPLL	AR_CHKSTOP	AR_SOFT	AR_CORE	AR_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	AR_LVD27	AR_SWT	AR_LVD12_PD1	AR_LVD12_PD0
		W																
0xC3FE_4014	reserved																	

Table 79. MC_RGM Memory Map (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_4018	RGM_FESS / RGM_STDBY	R	SS_EXR	0	0	0	0	0	0	SS_FLASH	SS_LVD45	SS_CMU_FHL	SS_CMU_OLR	SS_FMPLL	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	0	BOOT_FROM_BKP_RAM	0	0	0	0	0	0	0
		W																
0xC3FE_401C	RGM_FBRE	R	BE_EXR	0	0	0	0	0	0	BE_FLASH	BE_LVD45	BE_CMU_FHL	BE_CMU_OLR	BE_FMPLL	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
		W																
0xC3FE_4020 … 0xC3FE_7FFC	reserved																	

9.3.1 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_STDBY register may be accessed as a word at address 0xC3FE_4018, as a half-word at address 0xC3FE_401A, or as a byte at address 0xC3FE_401B.

Functional Event Status Register (RGM_FES)

Figure 76. Functional Event Status Register (RGM_FES)

Address 0xC3FE_4000 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	0	0	0	0	0	0	F_FLASH	F_LVD45	F_CMU_FHL	F_CMU_OLR	F_FMPLL	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
W	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. Register bits are cleared on write ‘1’.

Table 80. 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_FLASH	**Flag for code or data flash fatal error** 0 No code or data flash fatal error event has occurred since either the last clear or the last destructive reset assertion 1 A code or data flash fatal error event has occurred
F_LVD45	**Flag for 4.5 V low-voltage detected** 0 No 4.5 V low-voltage detected event has occurred since either the last clear or the last destructive reset assertion 1 A 4.5 V low-voltage detected event has occurred
F_CMU_FHL	**Flag for CMU clock frequency higher/lower than reference** 0 No CMU clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A CMU clock frequency higher/lower than reference event has occurred
F_CMU_OLR	**Flag for FXOSC frequency lower than reference** 0 No FXOSC frequency lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A FXOSC frequency lower than reference event has occurred
F_FMPLL	**Flag for FMPLL fail** 0 No FMPLL fail event has occurred since either the last clear or the last destructive reset assertion 1 A FMPLL fail event has occurred

Table 80. Functional Event Status Register (RGM_FES) Field Descriptions (continued)

Field	Description
F_CHKSTOP	**Flag for checkstop reset** 0 No checkstop reset event has occurred since either the last clear or the last destructive reset assertion 1 A checkstop reset 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 debug control core reset** 0 No debug control core reset event has occurred since either the last clear or the last destructive reset assertion 1 A debug control core reset event has occurred; this event can only be asserted when the DBCR0[RST] field is set by an external debugger. See the "Debug Support" chapter of the core reference manual for more details.
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: Supervisor 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	0	0	0	F_LVD27	F_SWT	F_LVD12_PD1	F_LVD12_PD0
W	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. Register bits are cleared on write ‘1’.

Table 81. 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 (due to either a software clear or a low-voltage detection) 1 A power-on event has occurred
F_LVD27	**Flag for 2.7 V low-voltage detected** 0 No 2.7 V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7 V low-voltage detected event has occurred
F_SWT	**Flag for software watchdog timer** 0 No software watchdog timer event has occurred since either the last clear or the last power-on reset assertion 1 A software watchdog timer event has occurred
F_LVD12_PD1	**Flag for 1.2 V low-voltage detected (power domain #1)** 0 No 1.2 V low-voltage detected (power domain #1) event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2 V low-voltage detected (power domain #1) event has occurred
F_LVD12_PD0	**Flag for 1.2 V low-voltage detected (power domain #0)** 0 No 1.2 V low-voltage detected (power domain #0) event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2 V low-voltage detected (power domain #0) event has occurred

Note: The F_POR flag is automatically cleared on a 1.2 V low-voltage detected (power domain #0 or #1) 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_PD0, <register>F_LVD12_PD1, or <register>F_LVD27 flag is set on exiting the reset sequence. Therefore, if the F_POR, <register>F_LVD12_PD0, <register>F_LVD12_PD1, or <register>F_LVD27 flags are set on reset exit, software should interpret the reset cause as power-on.

Note: In contrast to all other reset sources, the 1.2 V low-voltage detected (power domain #0) event is captured on its deassertion. Therefore, the status bit F_LVD12_PD0 is also asserted on the reset's deassertion. In case an alternate event is selected, the SAFE mode or interrupt request are similarly asserted on the reset's deassertion.

Functional Event Reset Disable Register (RGM_FERD)

Figure 78. Functional Event Reset Disable Register (RGM_FERD)

Address 0xC3FE_4004 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D_EXR	0	0	0	0	0	0	D_FLASH	D_LVD45	D_CMU_FHL	D_CMU_OLR	D_FMPLL	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.

Table 82. 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 1 An external reset event generates a **SAFE** mode request
D_FLASH	**Disable code or data flash fatal error** 0 A code or data flash fatal error event triggers a reset sequence 1 A code or data flash fatal error event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_FLASH**
D_LVD45	**Disable 4.5 V low-voltage detected** 0 A 4.5 V low-voltage detected event triggers a reset sequence 1 A 4.5 V low-voltage detected event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_LVD45**
D_CMU_FHL	**Disable CMU clock frequency higher/lower than reference** 0 A CMU clock frequency higher/lower than reference event triggers a reset sequence 1 A CMU clock frequency higher/lower than reference event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CMU_FHL**
D_CMU_OLR	**Disable FXOSC frequency lower than reference** 0 A FXOSC frequency lower than reference event triggers a reset sequence 1 A FXOSC frequency lower than reference event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CMU_OLR**
D_FMPLL	**Disable FMPLL fail** 0 A FMPLL fail event triggers a reset sequence 1 A FMPLL fail event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_FMPLL**

Table 82. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions (continued)

Field	Description
D_CHKSTOP	**Disable checkstop reset** 0 A checkstop reset event triggers a reset sequence 1 A checkstop reset event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_CHKSTOP**
D_SOFT	**Disable software reset** 0 A software reset event triggers a reset sequence 1 A software reset event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_FEAR.AR_SOFT**
D_CORE	**Disable debug control core reset** 0 A debug control core reset event triggers a reset sequence 1 A debug control core reset 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: 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	D_LVD27	D_SWT	D_LVD12_PD1	D_LVD12_PD0
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. When a destructive reset source is disabled, the associated destructive event will trigger either a safe mode request or an interrupt request (see *Section Destructive Event Alternate Request Register (RGM_DEAR)*).

Table 83. Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions

Field	Description
D_LVD27	**Disable 2.7 V low-voltage detected** 0 A 2.7 V low-voltage detected event triggers a reset sequence 1 A 2.7 V low-voltage detected event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_DEAR.AR_LVD27**
D_SWT	**Disable software watchdog timer** 0 A software watchdog timer event triggers a reset sequence 1 A software watchdog timer event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_DEAR**.
D_LVD12_PD1	**Disable 1.2 V low-voltage detected (power domain #1)** 0 A 1.2 V low-voltage detected (power domain #1) event triggers a reset sequence 1 A 1.2 V low-voltage detected (power domain #1) event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_DEAR.AR_LVD12_PD1**
D_LVD12_PD0	**Disable 1.2 V low-voltage detected (power domain #0)** 0 A 1.2 V low-voltage detected (power domain #0) event triggers a reset sequence 1 A 1.2 V low-voltage detected (power domain #0) event generates either a **SAFE** mode or an interrupt request depending on the value of **RGM_DEAR.AR_LVD12_PD0**

Functional Event Alternate Request Register (RGM_FEAR)

Figure 80. Functional Event Alternate Request Register (RGM_FEAR)

Address 0xC3FE_4010 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AR_EXR	0	0	0	0	0	0	AR_FLASH	AR_LVD45	AR_CMU_FHL	AR_CMU_OLR	AR_FMPLL	AR_CHKSTOP	AR_SOFT	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 84. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions

Field	Description
AR_EXR	**Alternate Request for External Reset** 0 Generate a **SAFE** mode request on an **external reset** event if the reset is disabled 1 Generate an interrupt request on an **external reset** event if the reset is disabled
AR_FLASH	**Alternate Request for code or data flash fatal error** 0 Generate a **SAFE** mode request on a code or data flash fatal error event if the reset is disabled 1 Generate an interrupt request on a code or data flash fatal error event if the reset is disabled
AR_LVD45	**Alternate Request for 4.5 V low-voltage detected** 0 Generate a **SAFE** mode request on a 4.5 V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 4.5 V low-voltage detected event if the reset is disabled
AR_CMU_FHL	**Alternate Request for CMU clock frequency higher/lower than reference** 0 Generate a **SAFE** mode request on a CMU clock frequency higher/lower than reference event if the reset is disabled 1 Generate an interrupt request on a CMU clock frequency higher/lower than reference event if the reset is disabled
AR_CMU_OLR	**Alternate Request for FXOSC frequency lower than reference** 0 Generate a **SAFE** mode request on a FXOSC frequency lower than reference event if the reset is disabled 1 Generate an interrupt request on a FXOSC frequency lower than reference event if the reset is disabled For the case when RGM_FERD[D_CMU_OLR] = 1 & RGM_FEAR[AR_CMU_OLR] = 1, an RGM interrupt will not be generated for an FXOSC failure when the system clock = FXOSC as there will be no system clock to execute the interrupt service routine. However, the interrupt service routine will be executed if the FXOSC recovers at some point. The recommended use case for this feature is when the system clock = FIRC or FMPLL.
AR_FMPLL	**Alternate Request for FMPLL fail** 0 Generate a **SAFE** mode request on a FMPLL fail event if the reset is disabled 1 Generate an interrupt request on a FMPLL fail event if the reset is disabled
AR_CHKSTOP	**Alternate Request for checkstop reset** 0 Generate a **SAFE** mode request on a checkstop reset event if the reset is disabled 1 Generate an interrupt request on a checkstop reset event if the reset is disabled
AR_SOFT	**Alternate Request for software reset** 0 Generate a **SAFE** mode request on a software reset event if the reset is disabled 1 Generate an interrupt request on a software reset event if the reset is disabled
AR_CORE	**Alternate Request for debug control core reset** 0 Generate a **SAFE** mode request on a debug control core reset event if the reset is disabled 1 Generate an interrupt request on a debug control core reset 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

Destructive Event Alternate Request Register (RGM_DEAR)

Figure 81. Destructive Event Alternate Request Register (RGM_DEAR)

Address 0xC3FE_4012 Access: 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	AR_LVD27	AR_SWT	AR_LVD12_PD1	AR_LVD12_PD0
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 destructive event has been disabled. The alternate request can be either a **SAFE** mode request to MC_ME or an interrupt request to the system.

Table 85. Destructive Event Alternate Request Register (RGM_DEAR) Field Descriptions

Field	Description
AR_LVD27	**Alternate Request for 2.7 V low-voltage detected** 0 Generate a **SAFE** mode request on a 2.7 V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 2.7 V low-voltage detected event if the reset is disabled
AR_SWT	**Alternate Request for software watchdog timer** 0 Generate a **SAFE** mode request on a software watchdog timer event if the reset is disabled 1 Generate an interrupt request on a software watchdog timer event if the reset is disabled
AR_LVD12_PD1	**Alternate Request for 1.2 V low-voltage detected (power domain #1)** 0 Generate a **SAFE** mode request on a 1.2 V low-voltage detected (power domain #1) event if the reset is disabled 1 Generate an interrupt request on a 1.2 V low-voltage detected (power domain #1) event if the reset is disabled
AR_LVD12_PD0	**Alternate Request for 1.2 V low-voltage detected (power domain #0)** 0 Generate a **SAFE** mode request on a 1.2 V low-voltage detected (power domain #0) event if the reset is disabled 1 Generate an interrupt request on a 1.2 V low-voltage detected (power domain #0) event if the reset is disabled

Functional Event Short Sequence Register (RGM_FESS)

Figure 82. Functional Event Short Sequence Register (RGM_FESS)

Address 0xC3FE_4018 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS_EXR	0	0	0	0	0	0	SS_FLASH	SS_LVD45	SS_CMU_FHL	SS_CMU_OLR	SS_FMPLL	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 86. 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** 1 The reset sequence triggered by an **external** reset event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**
SS_FLASH	**Short Sequence for code or data flash fatal error** 0 The reset sequence triggered by a code or data flash fatal error event will start from **PHASE1** 1 The reset sequence triggered by a code or data flash fatal error event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**
SS_LVD45	**Short Sequence for 4.5 V low-voltage detected** 0 The reset sequence triggered by a 4.5 V low-voltage detected event will start from **PHASE1** 1 The reset sequence triggered by a 4.5 V low-voltage detected event will start from PHASE3, skipping **PHASE1** and **PHASE2**
SS_CMU_FHL	**Short Sequence for CMU clock frequency higher/lower than reference** 0 The reset sequence triggered by a CMU clock frequency higher/lower than reference event will start from **PHASE1** 1 The reset sequence triggered by a CMU clock frequency higher/lower than reference event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**
SS_CMU_OLR	**Short Sequence for FXOSC frequency lower than reference** 0 The reset sequence triggered by a FXOSC frequency lower than reference event will start from **PHASE1** 1 The reset sequence triggered by a FXOSC frequency lower than reference event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**

Table 86. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions (continued)

Field	Description
SS_FMPLL	**Short Sequence for FMPLL fail** 0 The reset sequence triggered by a FMPLL fail event will start from **PHASE1** 1 The reset sequence triggered by a FMPLL fail event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**
SS_CHKSTOP	**Short Sequence for checkstop reset** 0 The reset sequence triggered by a checkstop reset event will start from **PHASE1** 1 The reset sequence triggered by a checkstop reset event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**
SS_SOFT	**Short Sequence for software reset** 0 The reset sequence triggered by a software reset event will start from **PHASE1** 1 The reset sequence triggered by a software reset event will start from **PHASE3**, skipping **PHASE1** and **PHASE2**
SS_CORE	**Short Sequence for debug control core reset** 0 The reset sequence triggered by a debug control core reset event will start from **PHASE1** 1 The reset sequence triggered by a debug control core reset 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**

STANDBY Reset Sequence Register (RGM_STDBY)

Figure 83. STANDBY Reset Sequence Register (RGM_STDBY)

Address 0xC3FE_401A Access: Supervisor 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	BOOT_FROM_BKP_RAM	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 defines the reset sequence to be applied on **STANDBY** mode exit. 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 87. STANDBY Reset Sequence Register (RGM_STDBY) Field Descriptions

Field	Description
BOOT_FROM_BKP_RAM	**Boot from Backup SRAM indicator** — This bit indicates whether the system will boot from backup SRAM or flash out of **STANDBY** exit. 0 Boot from default boot location on **STANDBY** exit 1 Boot from backup SRAM on **STANDBY** exit

Note: This register is reset on any enabled 'destructive' or 'functional' reset event.

Functional Bidirectional Reset Enable Register (RGM_FBRE)

Figure 84. Functional Bidirectional Reset Enable Register (RGM_FBRE)

Address 0xC3FE_401C Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BE_EXR	0	0	0	0	0	0	BE_FLASH	BE_LVD45	BE_CMU_FHL	BE_CMU_OLR	BE_FMPLL	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.

Table 88. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_EXR	**Bidirectional Reset Enable for External Reset** 0 RESET is asserted on an **external** reset event if the reset is enabled 1 RESET is not asserted on an **external** reset event
BE_FLASH	**Bidirectional Reset Enable for code or data flash fatal error** 0 RESET is asserted on a code or data flash fatal error event if the reset is enabled 1 RESET is not asserted on a code or data flash fatal error event
BE_LVD45	**Bidirectional Reset Enable for 4.5 V low-voltage detected** 0 RESET is asserted on a 4.5 V low-voltage detected event if the reset is enabled 1 RESET is not asserted on a 4.5 V low-voltage detected event
BE_CMU_FHL	**Bidirectional Reset Enable for CMU clock frequency higher/lower than reference** 0 RESET is asserted on a CMU clock frequency higher/lower than reference event if the reset is enabled 1 RESET is not asserted on a CMU clock frequency higher/lower than reference event
BE_CMU_OLR	**Bidirectional Reset Enable for FXOSC frequency lower than reference** 0 RESET is asserted on a FXOSC frequency lower than reference event if the reset is enabled 1 RESET is not asserted on a FXOSC frequency lower than reference event

Table 88. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions (continued)

Field	Description
BE_FMPLL	**Bidirectional Reset Enable for FMPLL fail** 0 RESET is asserted on a FMPLL fail event if the reset is enabled 1 RESET is not asserted on a FMPLL fail event
BE_CHKSTOP	**Bidirectional Reset Enable for checkstop reset** 0 RESET is asserted on a checkstop reset event if the reset is enabled 1 RESET is not asserted on a checkstop reset event
BE_SOFT	**Bidirectional Reset Enable for software reset** 0 RESET is asserted on a software reset event if the reset is enabled 1 RESET is not asserted on a software reset event
BE_CORE	**Bidirectional Reset Enable for debug control core reset** 0 RESET is asserted on a debug control core reset event if the reset is enabled 1 RESET is not asserted on a debug control core reset event
BE_JTAG	**Bidirectional Reset Enable for JTAG initiated reset** 0 RESET is asserted on a JTAG initiated reset event if the reset is enabled 1 RESET is not asserted on a JTAG initiated reset event

9.4 Functional Description

9.4.1 Reset State Machine

The main role of MC_RGM is the generation of the reset sequence which ensures that the correct parts of the device are reset based on the reset source event. This is summarized in *Table 89*.

Table 89. MC_RGM Reset Implications

Source	What Gets Reset	External Reset Assertion	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	yes	yes
'functional' resets	all except some clock/reset management and debug	programmable(1)	programmable(2)
shortened 'functional' resets(3)	flip-flops except some clock/reset management	programmable(1)	programmable(2)

1. the assertion of the external reset is controlled via the RGM_FBRE register
2. the boot mode is captured if the external reset is asserted
3. 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 85*.

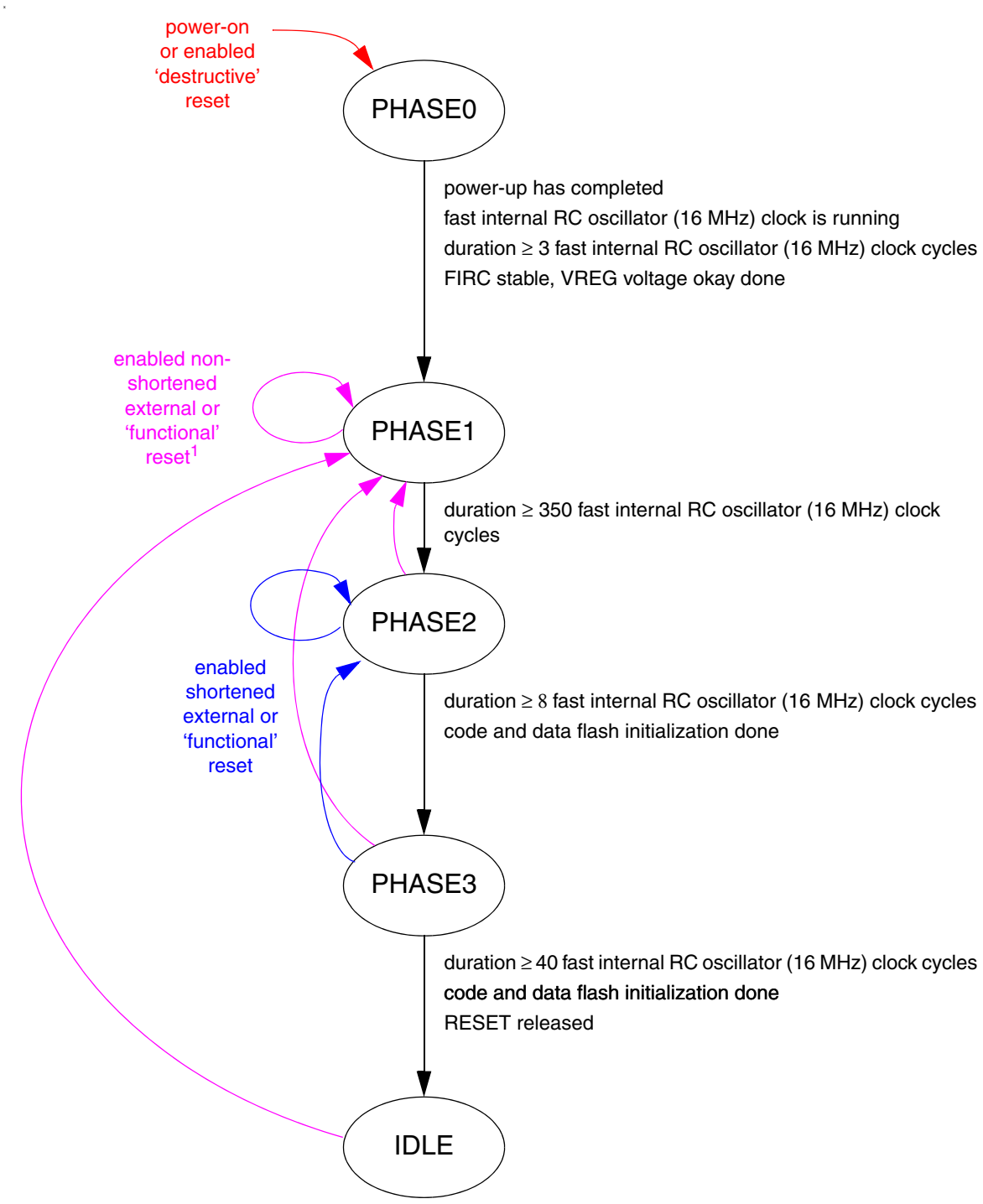


Figure 85. MC_RGM State Machine

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:

- power-up has completed
- fast internal RC oscillator (16 MHz) clock is running
- all enabled ‘destructive’ resets have been processed
- all processes that need to be done in PHASE0 are completed
  - FIRC stable, VREG voltage okay
- a minimum of 3 fast internal RC oscillator (16 MHz) 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 fast internal RC oscillator (16 MHz) 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 fast internal RC oscillator (16 MHz) 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 fast internal RC oscillator (16 MHz) 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 system to the platform and waits for new reset events that can trigger a reset sequence.

9.4.2 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 ‘destructive’ reset can be optionally disabled by writing bit RGM_DERD.D_<destructive reset>.

Note: The RGM_DERD register can be written only once between two power-on reset events.

The device’s low-voltage detector threshold ensures that, when 1.2 V low-voltage detected (power domain #0) 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.

9.4.3 External Reset

The MC_RGM manages the external reset coming from RESET. The detection of a falling edge on RESET will start the reset sequence from the beginning of PHASE1.

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.

9.4.4 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.

9.4.5 STANDBY Entry Sequence

STANDBY mode can be entered only when the MC_RGM is in IDLE. On STANDBY entry, the MC_RGM moves to PHASE1. The minimum duration counter in PHASE1 does not start until STANDBY mode is exited. On entry to PHASE1 due to STANDBY mode entry, the resets for all power domains except power domain #0 are asserted. During this time, RESET is not asserted as the external reset can act as a wakeup for the device.

There is an option to keep the flash inaccessible and in low-power mode on STANDBY exit by configuring the DRUN mode before STANDBY entry so that the flash is in power-down or low-power mode. If the flash is to be inaccessible, the PHASE2 and PHASE3 states do not wait for the flash to complete initialization before exiting, and the reset to the flash remains asserted.

See the MC_ME chapter for details on the STANDBY and DRUN modes.

9.4.6 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 each reset source event (except the power-on reset event) 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_F/DERD and RGM_F/DEAR registers as shown in *Table 90*.

Table 90. MC_RGM Alternate Event Selection

RGM_F/DERD Bit Value	RGM_F/DEAR 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_F/DES status bit.

Note: Alternate requests (SAFE mode as well as interrupt requests) are generated asynchronously.

Note: If a masked 'destructive' reset event which is configured to generate a SAFE mode/interrupt request occurs during PHASE0, it is ignored, and the MC_RGM will not send any safe mode/interrupt request to the MC_ME. The same is true for masked 'functional' reset events during PHASE1.

9.4.7 Boot Mode Capturing

The MC_RGM provides sampling of the boot mode PA[8] and PA[9] for use by the system to determine the boot mode. This sampling is done five fast internal RC oscillator (16 MHz) clock cycles before the rising edge of RESET. The result of the sampling is then provided to the system. For each bit, a value of '1' is produced only if each of the oldest three of the five samples have the value '1', otherwise a value of '0' is produced.

Note: In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value to the device at least five fast internal RC oscillator (16 MHz) clock periods before the external reset deassertion crosses the V_{IH} threshold.

Note: RESET can be low as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins.

10 Power Control Unit (MC_PCU)

10.1 Introduction

10.1.1 Overview

The power control unit (MC_PCU) is used to reduce the overall SoC power consumption. Power can be saved by disconnecting parts of the SoC from the power supply via a power switching device. The SoC is grouped into multiple parts having this capability which are called “power domains”.

When a power domain is disconnected from the supply, the power consumption is reduced to zero in that domain. Any status information of such a power domain is lost. When re-connecting a power domain to the supply voltage, the domain draws an increased current until the power domain reaches its operational voltage.

Power domains are controlled on a device mode basis. For each mode, software can configure whether a power domain is connected to the supply voltage (power-up state) or disconnected (power-down state). Maximum power saving is reached by entering the STANDBY mode.

On each mode change request, the MC_PCU evaluates the power domain settings in the power domain configuration registers and initiates a power-down or a power-up sequence for each individual power domain. The power-up/down sequences are handled by finite state machines to ensure a smooth and safe transition from one power state to the other.

Exiting the STANDBY mode can only be done via a system wakeup event as all power domains other than power domain #0 are in the power-down state.

In addition, the MC_PCU acts as a bridge for mapping the VREG peripheral to the MC_PCU address space.

Figure 86 depicts the MC_PCU block diagram.

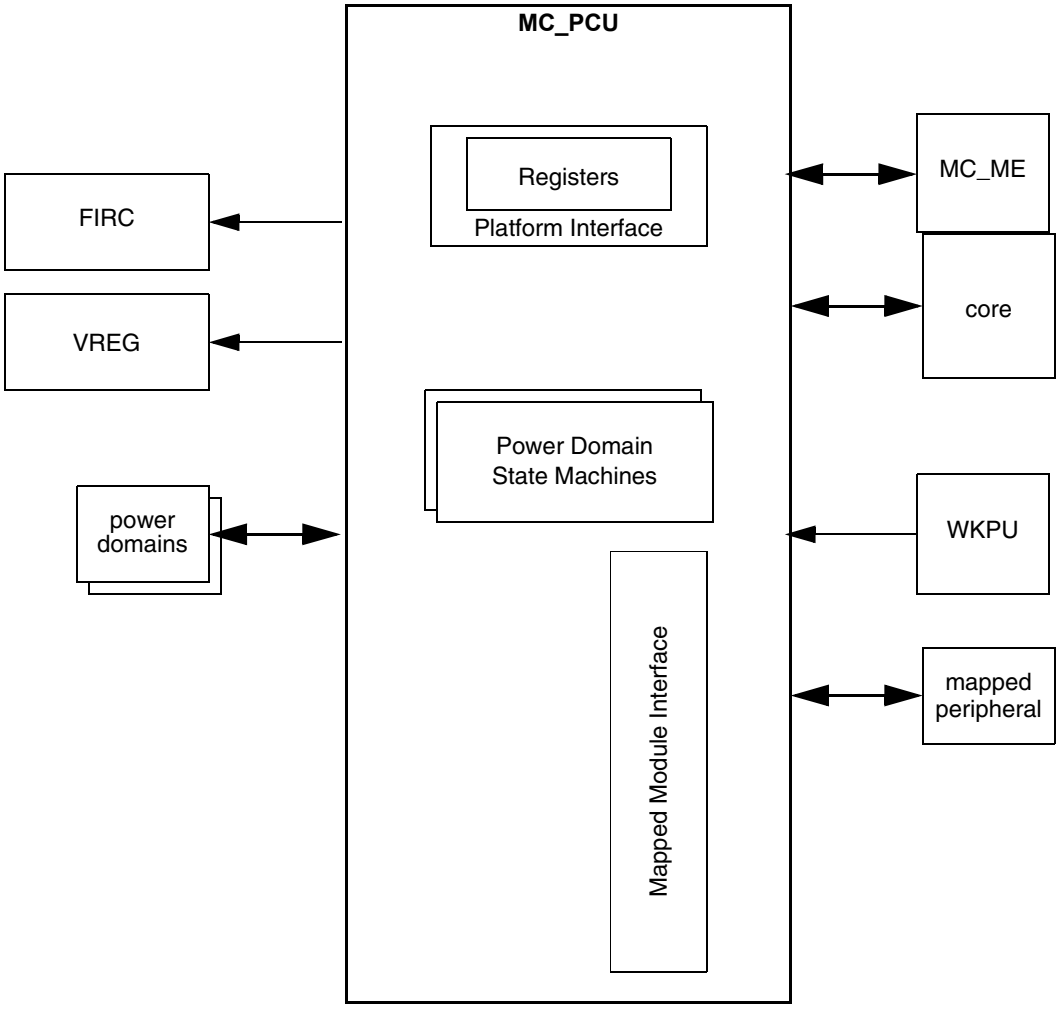


Figure 86. MC_PCU Block Diagram

10.1.2 Features

The MC_PCU includes the following features:

- support for 3 power domains
- support for device modes RESET, DRUN, SAFE, TEST, RUN0…3, HALT, STOP, and STANDBY (for further mode details, please see the MC_ME chapter)
- power states updating on each mode change and on system wakeup
- a handshake mechanism for power state changes thus guaranteeing operable voltage
- maps the VREG registers to the MC_PCU address space

10.1.3 Modes of Operation

The MC_PCU is available in all device modes.

10.2 External Signal Description

The MC_PCU has no connections to any external pins.

10.3 Memory Map and Register Definition

Table 91. MC_PCU Register Description

Address	Name	Description	Size	Access	Location
				Supervisor	
0xC3FE_8000	PCU_PCONF0	Power Domain #0 Configuration	word	read	*on page 10-222*
0xC3FE_8004	PCU_PCONF1	Power Domain #1 Configuration	word	read	*on page 10-224*
0xC3FE_8008	PCU_PCONF2	Power Domain #2 Configuration	word	read/write	*on page 10-224*
0xC3FE_8040	PCU_PSTAT	Power Domain Status Register	word	read	*on page 10-225*

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 92. MC_PCU Memory Map

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	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_8004	PCU_PCONF1	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_8008	PCU_PCONF2	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_800C … 0xC3FE_803C	reserved																	

Table 92. MC_PCU Memory Map (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_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	PD2	PD1	PD0
		W																
0x044 … 0x07C	reserved																	
0xC3FE_8080 … 0xC3FE_80FC	VREG registers																	
0xC3FE_8100 … 0xC3FE_BFFC	reserved																	

10.3.1 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 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 87. Power Domain #0 Configuration Register (PCU_PCONF0)

Address 0xC3FE_8000 Access: Supervisor 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	STOP	0	HALT	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 device 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 93. 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
RUN3	Power domain control during RUN3 mode 0 Power domain off 1 Power domain on
HALT	Power domain control during HALT mode 0 Power domain off 1 Power domain on
STOP	Power domain control during STOP mode 0 Power domain off 1 Power domain on
STBY0	Power domain control during STANDBY mode 0 Power domain off 1 Power domain on

Power Domain #1 Configuration Register (PCU_PCONF1)

Figure 88. Power Domain #1 Configuration Register (PCU_PCONF1)

Address 0xC3FE_8004 Access: Supervisor 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	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1

This register defines for power domain #1 whether it is on or off in each device mode. The bit field description is the same as in *Table 93*. As the platform, clock generation, and mode control reside in power domain #1, this power domain is only powered down during the STANDBY mode. Therefore, none of the bits is programmable. This register is available for completeness reasons.

The difference between PCU_PCONF0 and PCU_PCONF1 is the reset value of the STBY0 bit: During the STANDBY mode, power domain #1 is disconnected from the power supply, and therefore PCU_PCONF1.STBY0 is always ‘0’. Power domain #0 is always on, and therefore PCU_PCONF0.STBY0 is ‘1’.

For further details about STANDBY mode, please see *Section STANDBY Mode Transition*.

Power Domain #2 Configuration Register (PCU_PCONF2)

Figure 89. Power Domain #2 Configuration Register (PCU_PCONF2)

Address 0xC3FE_8008 Access: Supervisor 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	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1

This register defines for power domain #2 whether it is on or off in each device mode. The bit field description is the same as in *Table 93*.

Power Domain Status Register (PCU_PSTAT)

Figure 90. Power Domain Status Register (PCU_PSTAT)

Address 0xC3FE_8040 Access: Supervisor 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														PD2	PD1	PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

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

Table 94. Power Domain Status Register (PCU_PSTAT) Field Descriptions

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

10.4 Functional Description

10.4.1 General

The MC_PCU controls all available power domains on a device mode basis. The PCU_PCONFn registers specify during which system/user modes a power domain is powered up. The power state for each individual power domain is reflected by the bits in the PCU_PSTAT register.

On a mode change, the MC_PCU evaluates which power domain(s) must change power state. The power state is controlled by a state machine (FSM) for each individual power domain which ensures a clean and safe state transition.

10.4.2 Reset / Power-On Reset

After any reset, the SoC will transition to the RESET mode during which all power domains are powered up (see the MC_ME chapter). Once the reset sequence has been completed, the DRUN mode is entered and software can begin the MC_PCU configuration.

10.4.3 MC_PCU Configuration

Per default, all power domains are powered in all modes other than STANDBY. Software can change the configuration for each power domain on a mode basis by programming the PCU_PCONFn registers.

Each power domain which is powered down is held in a reset state. Read/write accesses to peripherals in those power domains will result in a transfer error.

10.4.4 Mode Transitions

On a mode change requested by the MC_ME, the MC_PCU evaluates the power configurations for all power domains. It compares the settings in the PCU_PCONFn registers for the new mode with the settings for the current mode. If the configuration for a power domain differs between the modes, a power state change request is generated. These requests are handled by a finite state machine to ensure a smooth and safe transition from one power state to another.

DRUN, SAFE, TEST, RUN0…3, HALT, and STOP Mode Transition

The DRUN, SAFE, TEST, RUN0…3, HALT, and STOP modes allow an increased power saving. The level of power saving is software-controllable via the settings in the PCU_PCONFn registers for power domain #2 onwards. The settings for power domains #0 and #1 can not be changed. Therefore, power domains #0 and #1 remain connected to the power supply for all modes beside STANDBY.

Figure 91 shows an example for a mode transition from RUN0 to HALT and back, which will result in power domain #2 being powered down during the HALT mode. In this case, PCU_PCONF2.HALT is programmed to be '0'.

When the MC_PCU receives the mode change request to HALT mode, it starts its power-down phase. During the power-down phase, clocks are disabled and the reset is asserted resulting in a loss of all information for this power domain.

Then the power domain is disconnected from the power supply (power-down state).

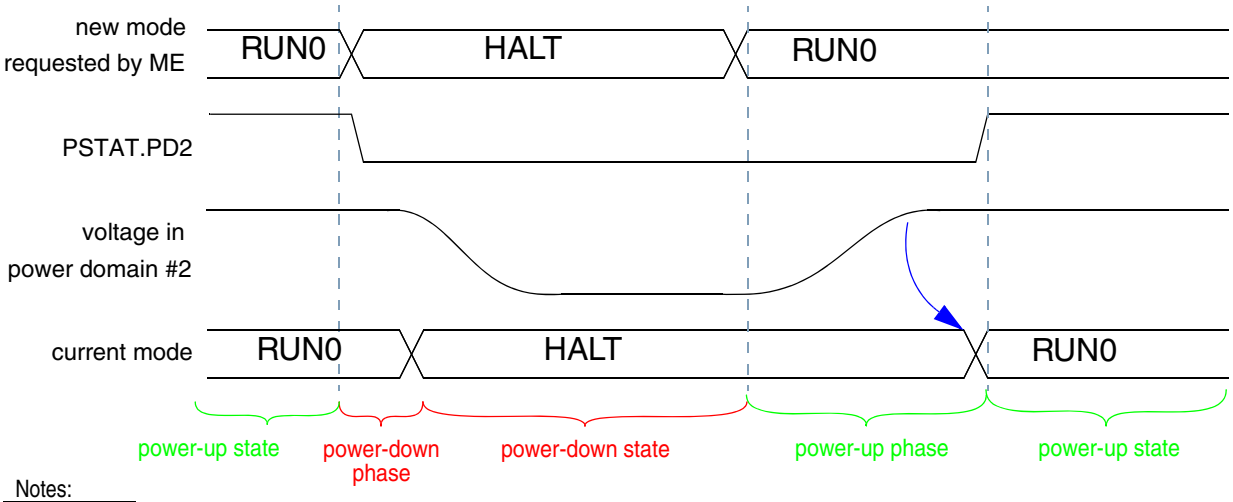


Figure 91. MC_PCU Events During Power Sequences (non-STANDBY mode)

When the MC_PCU receives a mode change request to RUN0, it starts its power-up phase if PCU_PCONF2.RUN0 is '1'. The power domain is re-connected to the power supply, and the voltage in power domain #2 will increase slowly. Once the voltage of power domain #2 is

within an operable range, its clocks are enabled, and its resets are deasserted (power-up state).

Note: It is possible that, due to a mode change, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.

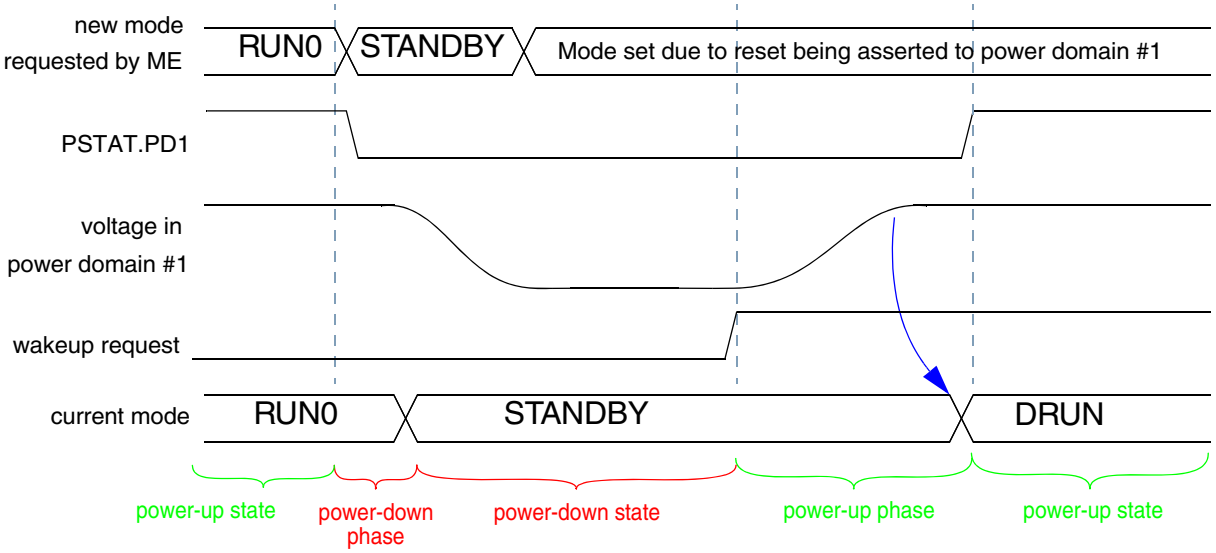
STANDBY Mode Transition

STANDBY offers the maximum power saving. The level of power saving is software-controllable via the settings in the PCU_PCONFn registers for power domain #2 onwards. Power domain #0 stays connected to the power supply while power domain #1 is disconnected from the power supply. Amongst others power domain #1 contains the platform and the MC_ME. Therefore this mode differs from all other user/system modes.

Once STANDBY is entered it can only be left via a system wakeup. On exiting the STANDBY mode, all power domains are powered up according to the settings in the PCU_PCONFn registers, and the DRUN mode is entered. In DRUN mode, at least power domains #0 and #1 are powered.

Figure 92 shows an example for a mode transition from RUN0 to STANDBY to DRUN. All power domains which have PCU_PCONFn.STBY0 cleared will enter power-down phase. In this example only power domain #1 will be disabled during STANDBY mode.

When the MC_PCU receives the mode change request to STANDBY mode it starts the power down phase for power domain #1. During the power down phase, clocks are disabled and reset is asserted resulting in a loss of all information for this power domain. Then the power domain is disconnected from the power supply (power-down state).



Notes:

Not drawn to scale; PCONF1.RUN0 = 1; PCONF1.STBY0 = 0

Figure 92. MC_PCU Events During Power Sequences (STANDBY mode)

When the MC_PCU receives a system wakeup request, it starts the power-up phase. The power domain is re-connected to the power supply and the voltage in power domain #1 will

increase slowly. Once the voltage is in an operable range, clocks are enabled and the reset is be deasserted (power-up state).

Note: It is possible that due to a wakeup request, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.

Power Saving for Memories During STANDBY Mode

All memories which are not powered down during STANDBY mode automatically enter a power saving state. No software configuration is required to enable this power saving state. While a memory is residing in this state an increased power saving is achieved. Data in the memories is retained.

10.5 Initialization Information

To initialize the MC_PCU, the registers PCU_PCONF2… should be programmed. After programming is done, those registers should no longer be changed.

10.6 Application Information

10.6.1 STANDBY Mode Considerations

STANDBY offers maximum power saving possibility. But power is only saved during the time a power domain is disconnected from the supply. Increased power is required when a power domain is re-connected to the power supply. Additional power is required during restoring the information (e.g. in the platform). Care should be taken that the time during which the SoC is operating in STANDBY mode is significantly longer than the required time for restoring the information.

11 Voltage Regulators and Power Supplies

11.1 Voltage regulators

The power blocks provide a 1.2 V digital supply to the internal logic of the device. The main supply is (3.3 V–5 V ± 10%) and digital/regulated output supply is (1.2 V ± 10%). The voltage regulator used in SPC560Bx and SPC560Cx comprises three regulators.

- High power regulator (HPREG)
- Low power regulator (LPREG)
- Ultra low power regulator (ULPREG)

The HPREG and LPREG regulators are switched off during STANDBY mode to save consumption from the regulator itself. In STANDBY mode, the supply is provided by the ULPREG regulator.

In STOP mode, the user can configure the HPREG regulator to switch-off (Refer to MC_ME chapter). In this case, when current is low enough to be handled by LPREG alone, the HPREG regulator is switch-off and the supply is provided by the LPREG regulator.

The internal voltage regulator requires an 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 to the associated pins.

The regulator has two digital domains, one for the high power regulator (HPREG) and the low power regulator (LPREG) called “High Power domain” and another one for the ultra low power regulator (ULPREG) called “Standby domain.” For each domain there is 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 thresholds, one at the 3.3 V level and the other one at the 5 V level.

11.1.1 High power regulator (HPREG)

The HPREG converts the 3.3 V–5 V input supply to a 1.2 V digital supply. For more information, see the voltage regulator electrical characteristics section of the datasheet.

The regulator can be switched off by software. Refer to the main voltage regulator control bit (MVRON) of the mode configuration registers in the mode entry module chapter of the reference manuals.

11.1.2 Low power regulator (LPREG)

The LPREG generates power for the device in the STOP mode, providing the output supply of 1.2 V. It always sees the minimum external capacitance. The control part of the regulator can be used to disable the low power regulator. It is managed by MC_ME.

11.1.3 Ultra low power regulator (ULPREG)

The ULPREG generates power for the standby domain as well as a part of the main domain and might or might not see the external capacitance. The control circuit of ULPREG can be used to disable the ultra low power regulator by software: This action is managed by MC_ME.

11.1.4 LVDs and POR

There are three kinds of LVD available:

1. LVD_MAIN for the 3.3 V–5 V input supply with thresholds at approximately 3 V level[h]
2. LVD_MAIN5 for the 3.3 V–5 V input supply with threshold at approximately 4.5 V level[h]
3. LVD_DIG for the 1.2 V output voltage

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.

Two LVD_DIGs are provided in the design. One LVD_DIG is placed in the high power domain and senses the HPREG/LPREG output notifying that the 1.2 V output is stable. The other LVD_DIG is placed in the standby domain and senses the standby 1.2 V supply level notifying that the 1.2 V output is stable. The reference voltage used for all LVDs is generated by the low power reference generator and is trimmed for LVD_DIG, using the bits LP[4:7]. Therefore, during the pre-trimming period, LVD_DIG exhibits higher thresholds, whereas during post trimming, the thresholds come in the desired range. Power-down pins are provided for LVDs. When LVDs are power-down, their outputs are pulled high.

POR is required to initialize the device during supply rise. POR works only on the rising edge of the 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 Vdd supply is above V_{PORUP} min (refer to datasheet for details). It will be released only after Vdd supply is above V_{PORH} (refer to datasheet for details). Vdd above V_{PORH} ensures power management module including internal LVDs modules are fully functional.

11.1.5 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 Ultra Low Power domain is powered, is used at power-up to release reset to temporization counter. At exit from low-power modes, the power-down for high power regulator request signal is monitored by the digital interface and used to release reset to the temporization counter. In both cases, 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 MC_RGM to release the reset to the device. It manages other specific requirements, like the transition between high power/low power mode to ultra low power mode avoiding a voltage drop below the permissible threshold limit of 1.08 V.

The VREG digital interface also holds control register to mask 5 V LVD status coming from the voltage regulator at the power-up.

11.1.6 Register description

The VREG_CTL register is mapped to the MC_PCU address space as described in *10, Power Control Unit (MC_PCU)*.

h. See section “Voltage monitor electrical characteristics” of the datasheet for detailed information about this voltage value.

Figure 93. 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	0	0	0	0	0	0	0	0	5V_LVD_MASK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 95. 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 MC_RGM for 5 V trip. 1: 5 V LVD is masked 0: 5 V LVD is not masked.

11.2 Power supply strategy

From a power-routing perspective, the device is organized as follows.

The device provides four dedicated supply domains at package level:

1. HV (high voltage external power supply for I/Os and most analog module) — This must be provided externally through VDD_HV/VSS_HV power pins. Voltage values should be aligned with V_{DD}/V_{SS}. Refer to datasheet for details.
2. ADC (high voltage external power supply for ADC module) — This must be provided externally through VDD_HV_ADC/VSS_HV_ADC power pins. Voltage values should be aligned with $V_{DD\_HV\_ADC}/V_{SS\_HV\_ADC}$. Refer to datasheet for details.
3. BV (high voltage external power supply for voltage regulator module) — This must be provided externally through VDD_BV_/VSS_BV power pins. Voltage values should be aligned with V_{DD}/V_{SS}. Refer to datasheet for details.
4. LV (low voltage internal power supply for core, FMPLL and Flash digital logic) — This is generated internally by embedded voltage regulator and provided to the core, FMPLL and Flash. Three VDD_LV/VSS_LV pins pairs are provided to connect the three decoupling capacitances. This is generated internally by internal voltage regulator but provided outside to connect stability capacitor. Refer to datasheet for details.

The four dedicated supply domains are further divided within the package in order to reduce as much as possible EMC and noise issues.

- HV_IO: High voltage pad supply
- HV_FLAn: High voltage Flash supply
- HV_OSC0REG[i]: High voltage external oscillator and regulator supply
- HV_ADR: High voltage reference for ADC module. Supplies are further star routed to reduce impact of ADC resistive reference on ADC capacitive reference accuracy.
- HV_ADV: High voltage supply for ADC module
- BV: High voltage supply for voltage regulator ballast. These two ballast pads are used to supply core and Flash. Each pad contains two ballasts to supply 80 mA and 20 mA respectively. Core is hence supplied through two ballasts of 80 mA capability and CFlash and DFlash through two 20 mA ballasts. The HV supply for both ballasts is shorted through double bonding.
- LV_COR: Low voltage supply for the core. It is also used to provide supply for FMPLL through double bonding.
- LV_FLAn: Low voltage supply for Flash module n. It is supplied with dedicated ballast and shorted to LV_COR through double bonding.
- LV_PLL[j]: Low voltage supply for FMPLL

11.3 Power domain organization

Based on stringent requirements for current consumption in different operational modes, the device is partitioned into different power domains. Organization into these power domains primarily means separate power supplies which are separated from each other by use of power switches (switch SW1 for power domain No. 1 and switch SW2 for power domain No. 2 as shown in *Figure 94*). These different separated power supplies are hence enabling to switch off power to certain regions of the device to avoid even leakage current consumption in logic supplied by the corresponding power supply.

This device employs three primary power domains, namely PD0, PD1 and PD2.

As PCU supports dynamic power down of domains based on different device mode, such a possible domain is depicted below in dotted periphery.

Power domain organization and connections to the internal regulator are depicted in *Figure 94*.

i. Regulator ground is separated from oscillator ground and shorted to the LV ground through star routing

j. During production test, it is also possible to provide the VDD_LV externally through pins by configuring regulator in bypass mode.

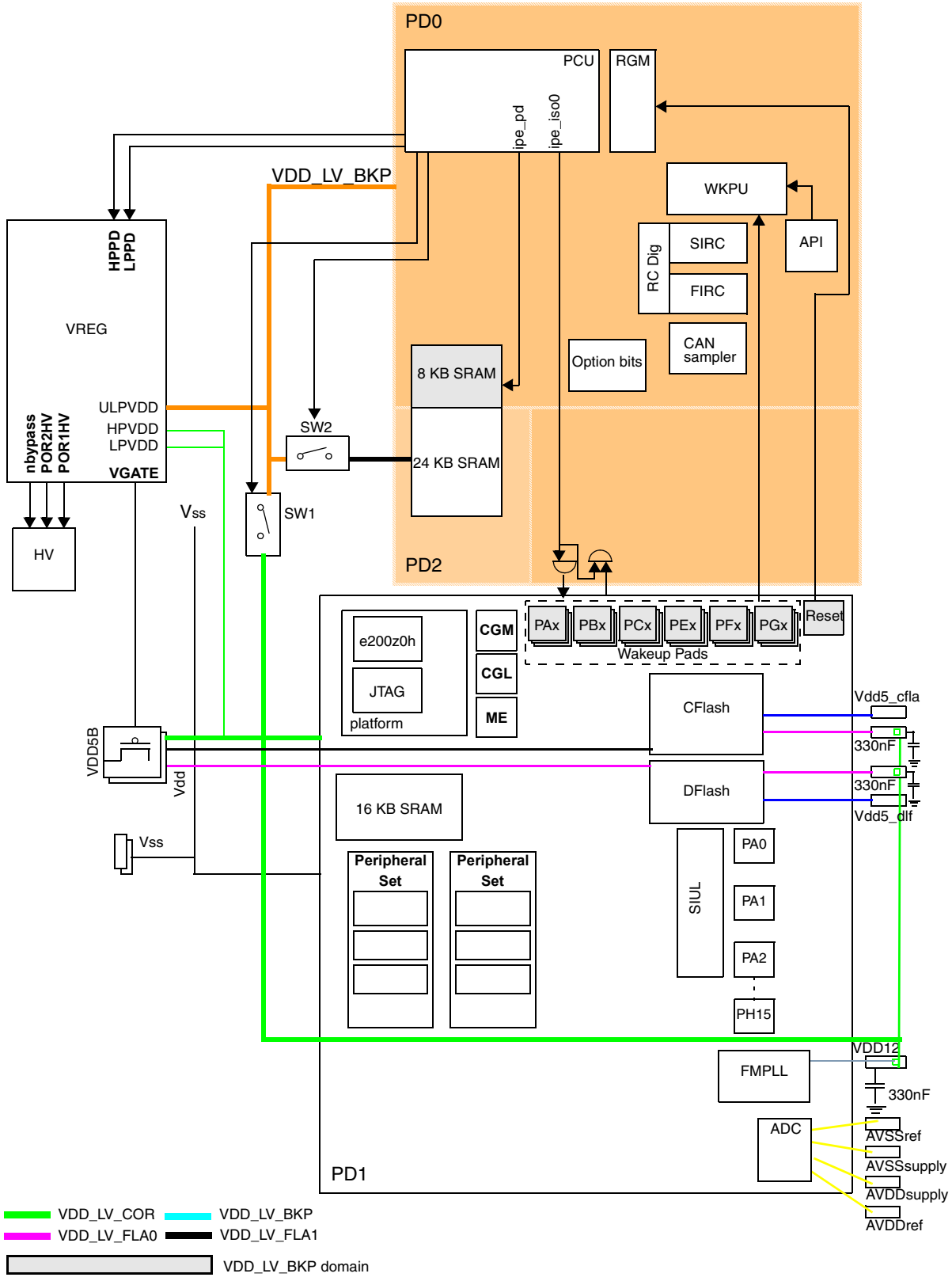


Figure 94. Power domain organization

12 Wakeup Unit (WKPU)

12.1 Overview

The Wakeup Unit supports 2 internal sources (WKPU[0:1]) and up to 18[k] external sources (WKPU[2:19]) that can generate interrupts or wakeup events, of which 1 can cause non-maskable interrupt requests. *Figure 95* is the block diagram of the Wakeup Unit and its interfaces to other system components.

The wakeup vector mapping is shown in *Table 96*. All unused WKPU pins must use a pull resistor — either pullup (internal or external) or pulldown (external) — to ensure no leakage from floating inputs.

Table 96. Wakeup vector mapping

Wakeup number	Port	SIU PCR#	Port input function[1] (can be used in conjunction with WKPU function)	WKPU IRQ to INTC	IRQ#	WISR	Register[2] bit position	Package			
								64-pin QFP	100-pin QFP	144-pin QFP	208-pin BGA
WKPU0	API	n/a[3]	—	WakeUp_IRQ_0	46	EIF0	31	✓[3]	✓[3]	✓[3]	✓[3]
WKPU1	RTC	n/a[3]	—			EIF1	30	✓[3]	✓[3]	✓[3]	✓[3]
WKPU2	PA1	PCR1	NMI			EIF2	29	✓	✓	✓	✓
WKPU3	PA2	PCR2	—			EIF3	28	✓	✓	✓	✓
WKPU4	PB1	PCR17	CAN0-RX			EIF4	27	✓	✓	✓	✓
WKPU5	PC11	PCR43	CAN1-RX, CAN4-RX			EIF5	26	x[4]	✓	✓	✓
WKPU6	PE0	PCR64	CAN5-RX			EIF6	25	x[4]	✓	✓	✓
WKPU7	PE9	PCR73	CAN2-RX, CAN3-RX			EIF7	24	x[4]	✓	✓	✓
WKPU8	PB10	PCR26	—	WakeUp_IRQ_1	47	EIF8	23	✓	✓	✓	✓
WKPU9	PA4	PCR4	—			EIF9	22	✓	✓	✓	✓
WKPU10	PA15	PCR15	—			EIF10	21	✓	✓	✓	✓
WKPU11	PB3	PCR19	LIN0-RX			EIF11	20	✓	✓	✓	✓
WKPU12	PC7	PCR39	LIN1-RX			EIF12	19	✓	✓	✓	✓
WKPU13	PC9	PCR41	LIN2-RX			EIF13	18	✓	✓	✓	✓
WKPU14	PE11	PCR75	LIN3-RX			EIF14	17	x[4]	✓	✓	✓
WKPU15	PF11	PCR91	—			EIF15	16	x[4]	x[4]	✓	✓

k. Up to 18 external sources in 144-pin LQFP and 208BGA; up to 14 external sources in 100-pin LQFP

Table 96. Wakeup vector mapping (continued)

Wakeup number	Port	SIU PCR#	Port input function[1] (can be used in conjunction with WKPU function)	WKPU IRQ to INTC	IRQ#	WISR	Register [2] bit position	Package			
								64-pin QFP	100-pin QFP	144-pin QFP	208-pin BGA
WKPU16	PF13	PCR93	—	WakeUp_IRQ_2	48	EIF16	15	x[4]	x[4]	✓	✓
WKPU17	PG3	PCR99	—			EIF17	14	x[4]	x[4]	✓	✓
WKPU18	PG5	PCR101	—			EIF18	13	x[4]	x[4]	✓	✓
WKPU19	PA0	PCR0	—			EIF19	12	✓	✓	✓	✓

1. This column does not contain an exhaustive list of functions on that pin. Rather, it includes peripheral communication functions (such as CAN and LINFlex Rx) that could be used to wake up the microcontroller. DSPI pins are not included because DSPI would typically be used in master mode.
2. WISR, IRER, WRER, WIFEER, WIFEEF, WIFER, WIPUER
3. Port not required to use timer functions.
4. Unavailable WKPU pins must use internal pullup enabled using WIPUER.

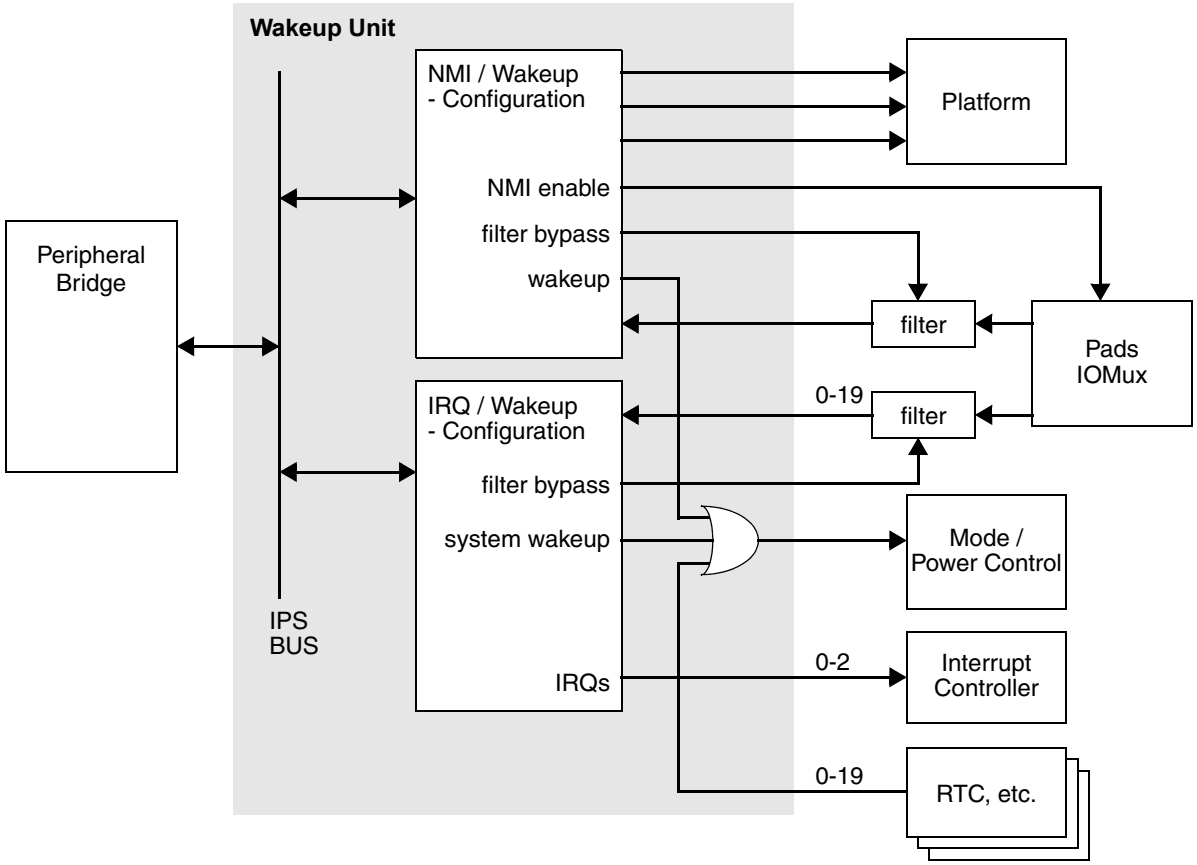


Figure 95. WKPU block diagram

12.2 Features

The Wakeup Unit supports these distinctive features:

- Non-maskable interrupt support with
  - 1 NMI source with bypassable glitch filter
  - Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
  - Edge detection
- External wakeup/interrupt support with
  - 3 system interrupt vectors for up to 18 interrupt sources
  - Analog glitch filter per each wakeup line
  - Independent interrupt mask
  - Edge detection
  - Configurable system wakeup triggering from all interrupt sources
  - Configurable pullup
- On-chip wakeup support
  - 2 wakeup sources
  - Wakeup status mapped to same register as external wakeup/interrupt status

12.3 External signal description

The Wakeup Unit has 18 signal inputs that can be used as external interrupt sources in normal RUN mode or as system wakeup sources in all power down modes.

The 18 external signal inputs include one signal input that can be used as a non-maskable interrupt source in normal RUN, HALT or STOP modes or a system wakeup source in STOP or STANDBY modes.

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 WKPU_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.

12.4 Memory map and register description

This section provides a detailed description of all registers accessible in the WKPU module.

12.4.1 Memory map

Table 97 gives an overview on the WKPU registers implemented.

Table 97. WKPU memory map

Base address: 0xC3F9_4000		
Address offset	**Register name**	**Location**
0x00	NMI Status Flag Register (NSR)	*on page 12-237*
0x04 – 0x07	Reserved	
0x08	NMI Configuration Register (NCR)	*on page 12-238*
0x0C – 0x13	Reserved	
0x14	Wakeup/Interrupt Status Flag Register (WISR)	*on page 12-239*
0x18	Interrupt Request Enable Register (IRER)	*on page 12-240*
0x1C	Wakeup Request Enable Register (WRER)	*on page 12-240*
0x20 – 0x27	Reserved	
0x28	Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)	*on page 12-241*
0x2C	Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)	*on page 12-241*
0x30	Wakeup/Interrupt Filter Enable Register (WIFER)	*on page 12-241*
0x34	Wakeup/Interrupt Pullup Enable Register (WIPUER)	*on page 12-242*

Note: Reserved registers will read as 0, writes will have no effect. If SSCM_ERROR[RAE] is enabled, a transfer error will be issued when trying to access completely reserved register space.

12.4.2 NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Figure 96. NMI Status Flag Register (NSR)

Offset: 0x00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NIF0	NOVF0	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 98. NSR field descriptions

Field	Description
NIF0	NMI Status Flag If enabled (NREE0 or NFEE0 set), NIF0 causes an interrupt request. 1 An event as defined by NREE0 and NFEE0 has occurred 0 No event has occurred on the pad
NOVF0	NMI Overrun Status Flag It will be a copy of the current NIF0 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 (NREE0 or NFEE0 set), NOVF0 causes an interrupt request. 1 An overrun has occurred on NMI input 0 No overrun has occurred on NMI input

12.4.3 NMI Configuration Register (NCR)

This register holds the configuration bits for the non-maskable interrupt settings.

Figure 97. NMI Configuration Register (NCR)

Offset: 0x08 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NLOCK0	NDSS0		NWRE0	0	NREE0	NFEE0	NFE0	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 99. NCR field descriptions

Field	Description
NLOCK0	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.
NDSS0	NMI Destination Source Select 00 Non-maskable interrupt 01 Critical interrupt 10 Machine check request 11 Reserved—no NMI, critical interrupt, or machine check request generated

Table 99. NCR field descriptions (continued)

Field	Description
NWRE0	NMI Wakeup Request Enable 1 A set NIF0 bit or set NOVF0 bit causes a system wakeup request 0 System wakeup requests from the corresponding NIF0 bit are disabled *Note: Software should only enable the NMI after the IVPR/IVOR registers have been configured. This should be noted when booting from RESET or STANDBY mode as all registers will have been cleared to their reset state.*
NREE0	NMI Rising-edge Events Enable 1 Rising-edge event is enabled 0 Rising-edge event is disabled
NFEE0	NMI Falling-edge Events Enable 1 Falling-edge event is enabled 0 Falling-edge event is disabled
NFE0	NMI Filter Enable Enable analog glitch filter on the NMI pad input. 1 Filter is enabled 0 Filter is disabled

Note: Writing a '0' to both NREE0 and NFEE0 disables the NMI functionality completely (that is, no system wakeup or interrupt will be generated on any pad activity)!

12.4.4 Wakeup/Interrupt Status Flag Register (WISR)

This register holds the wakeup/interrupt flags.

Figure 98. Wakeup/Interrupt Status Flag Register (WISR)

Offset: 0x14 Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	EIF[19:0](1)																			
W													w1c																			
Reset	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

1. EIF[18:15] are not available in all 100-pin packages.

Table 100. WISR field descriptions

Field	Description
EIF[x]	External Wakeup/Interrupt WKPU[x] Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request. 1 An event as defined by WIREER and WIFEER has occurred 0 No event has occurred on the pad

Note: Status bits associated with on-chip wakeup sources are located to the left of the external wakeup/interrupt status bits and are read only. The wakeup for these sources must be

configured and cleared at the on-chip wakeup source. Also, the configuration registers for the external interrupts/wakeups do not have corresponding bits.

12.4.5 Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging from the wakeup/interrupt pads to the interrupt controller.

Figure 99. Interrupt Request Enable Register (IRER)

Offset: 0x18 Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	EIRE[19:0](1)																			
W													w1c																			
Reset	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

1. EIRE[18:15] are not available in all 100-pin packages.

Table 101. IRER field descriptions

Field	Description
EIRE[x]	External Interrupt Request Enable x 1 A set EIF[x] bit causes an interrupt request 0 Interrupt requests from the corresponding EIF[x] bit are disabled

12.4.6 Wakeup Request Enable Register (WRER)

This register is used to enable the system wakeup messaging from the wakeup/interrupt pads to the mode entry and power control modules.

Figure 100. Wakeup Request Enable Register (WRER)

Offset: 0x1C Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	WRE[19:0](1)																			
W																																
Reset	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

1. WRE[18:15] are not available in all 100-pin packages.

Table 102. WRER field descriptions

Field	Description
WRE[x]	External Wakeup Request Enable x 1 A set EIF[x] bit causes a system wakeup request 0 System wakeup requests from the corresponding EIF[x] bit are disabled

12.4.7 Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)

This register is used to enable rising-edge triggered events on the corresponding wakeup/interrupt pads.

Note: The RTC_API can only be configured on the rising edge.

Figure 101. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)

Offset: 0x28 Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	IREE[19:0](1)																			
W																																
Reset	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

1. IREE[18:15] are not available in all 100-pin packages.

Table 103. WIREER field descriptions

Field	Description
IREE[x]	External Interrupt Rising-edge Events Enable x 1 Rising-edge event is enabled 0 Rising-edge event is disabled

12.4.8 Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)

This register is used to enable falling-edge triggered events on the corresponding wakeup/interrupt pads.

Figure 102. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)

Offset: 0x2C Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	IFEE[19:0](1)																			
W																																
Reset	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

1. IFEE[18:15] are not available in all 100-pin packages.

Table 104. WIFEER field descriptions

Field	Description
IFEEx	External Interrupt Falling-edge Events Enable x 1 Falling-edge event is enabled 0 Falling-edge event is disabled

12.4.9 Wakeup/Interrupt Filter Enable Register (WIFER)

This register is used to enable an analog filter on the corresponding interrupt pads to filter out glitches on the inputs.

Note: There is no analog filter for the RTC_API.

Figure 103. Wakeup/Interrupt Filter Enable Register (WIFER)

Offset: 0x30 Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	IFE[19:0](1)																			
W																																
Reset	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

1. IFE[18:15] are not available in all 100-pin packages.

Table 105. WIFER field descriptions

Field	Description
IFE[x]	External Interrupt Filter Enable x Enable analog glitch filter on the external interrupt pad input. 1 Filter is enabled 0 Filter is disabled

12.4.10 Wakeup/Interrupt Pullup Enable Register (WIPUER)

This register is used to enable a pullup on the corresponding interrupt pads to pull an unconnected wakeup/interrupt input to a value of ‘1’.

Figure 104. Wakeup/Interrupt Pullup Enable Register (WIPUER)

Offset: 0x34 Access: User read/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	0	0	0	0	0	0	0	0	0	0	0	0	IPUE[19:0](1)																			
W																																
Reset	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

1. IPUE[18:15] are not available in all 100-pin packages.

Table 106. WIPUER field descriptions

Field	Description
IPUE[x]	External Interrupt Pullup Enable x 1 Pullup is enabled 0 Pullup is disabled

12.5 Functional description

12.5.1 General

This section provides a complete functional description of the Wakeup Unit.

12.5.2 Non-maskable interrupts

The Wakeup Unit supports one non-maskable interrupt which is allocated to the following pins:

- 100-pin LQFP: Pin 7
- 144-pin LQFP: Pin 11
- 208-pin BGA: Pin F3

The Wakeup Unit supports the generation of three types of interrupts from the NMI. 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.

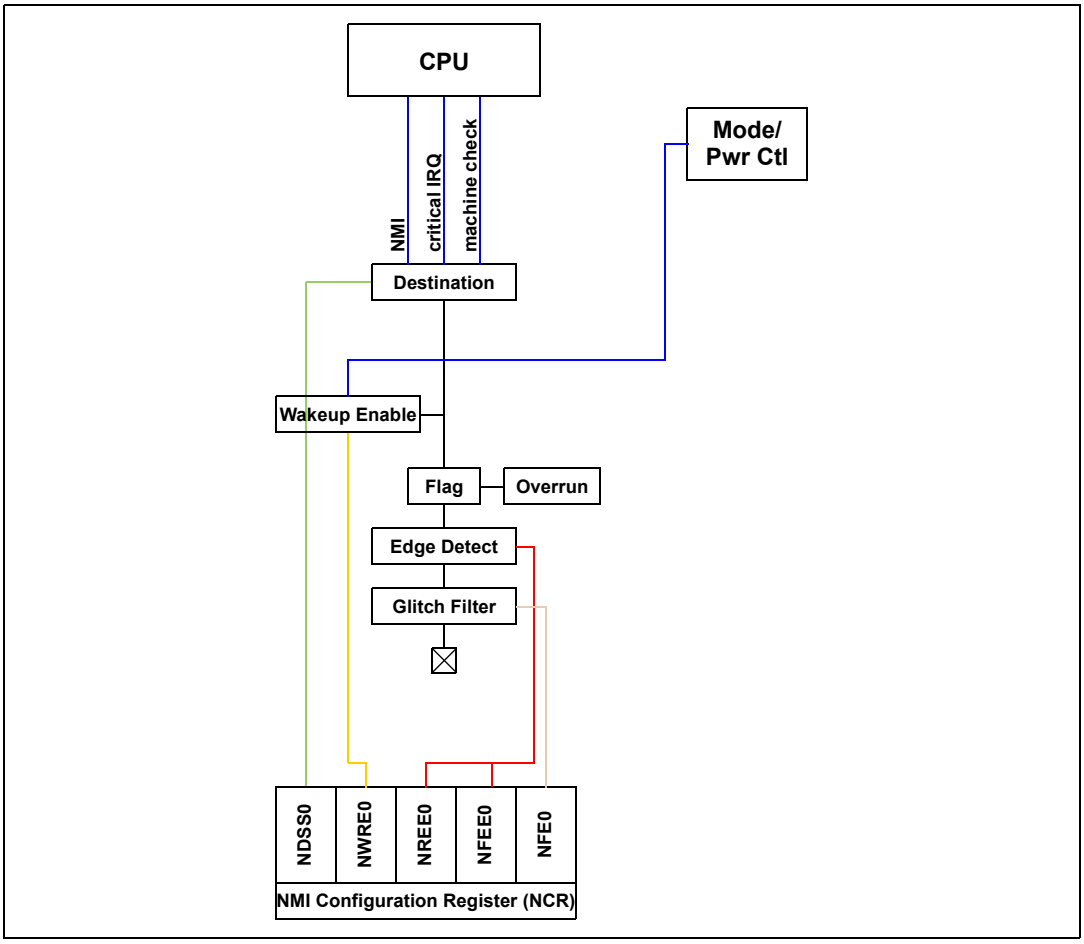


Figure 105. NMI pad diagram

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 97*). 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 NREE0 and NFEE0 bits.

Note: After reset, NREE0 and NFEE0 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 NDSS0 field. See *Table 99* for details.

An NMI supports a status flag and an overrun flag which are located in the NSR register (see *Figure 96*). The NIF0 and NOVF0 fields in this register are cleared by writing a '1' to them; this prevents inadvertent overwriting of other flags in the 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.

12.5.3 External wakeups/interrupts

The Wakeup Unit supports up to 18 external wakeup/interrupts which can be allocated to any pad necessary at the SoC level. This allocation is fixed per SoC.

The Wakeup Unit supports up to three interrupt vectors to the interrupt controller of the SoC. Each interrupt vector can support up to the number of external interrupt sources from the device pads with the total across all vectors being equal to the number of external interrupt sources. Each external interrupt source is assigned to exactly one interrupt vector. The interrupt vector assignment is sequential so that one interrupt vector is for external interrupt sources 0 through N-1, the next is for N through N+M-1, and so forth.

See *Figure 106* for an overview of the external interrupt implementation for the example of three interrupt vectors with up to eight external interrupt sources each.

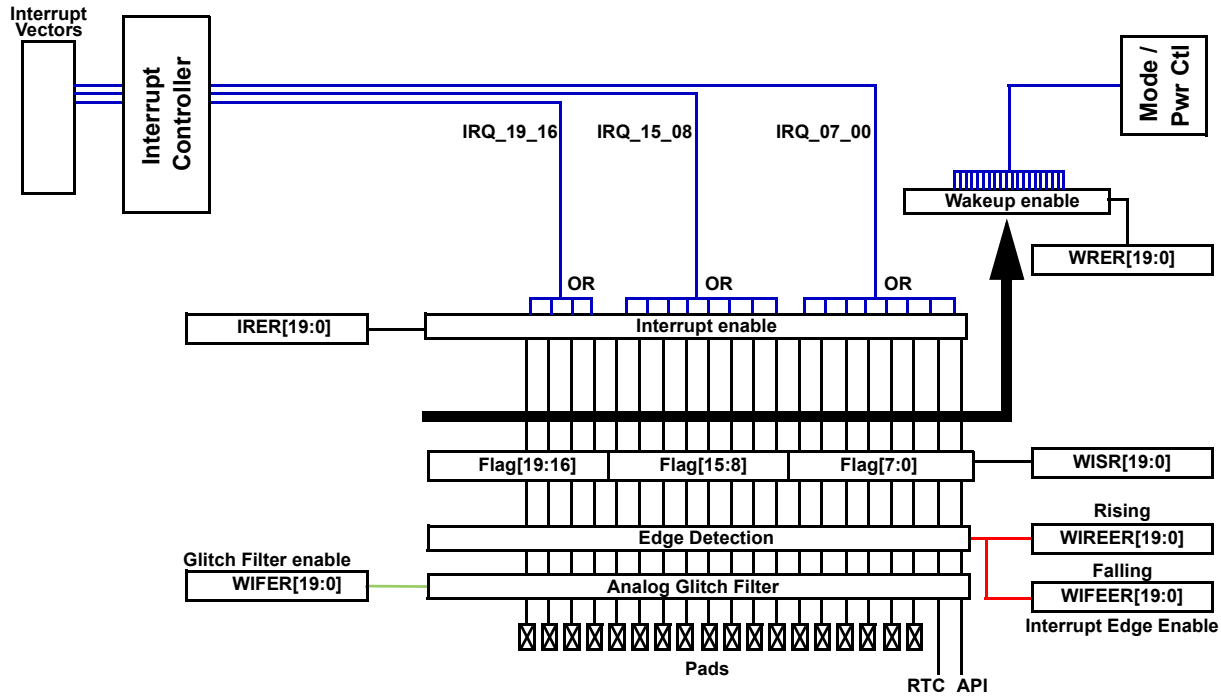


Figure 106. External interrupt pad diagram

All of the external interrupt pads within a single group have equal priority. It is the responsibility of the user software to search through the group of sources in the most appropriate way for their application.

Note: Glitch filter control and pad configuration should be done while the external interrupt line is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

External interrupt management

Each external interrupt can be enabled or disabled independently. This can be performed using a single rolled up register (*Figure 99*). A pad defined as an external interrupt can be configured by the user to recognize external interrupts with an active rising edge, an active falling edge or both edges being active.

Note: Writing a '0' to both IREE[x] and IFEE[x] disables the external interrupt functionality for that pad completely (that is, no system wakeup or interrupt will be generated on any activity on that pad)!

The active IRQ edge is controlled by the users through the configuration of the registers WIREER and WIFEER.

Each external interrupt supports an individual flag which is held in the flag register (WISR). The bits in the WISR[EIF] field are cleared by writing a '1' to them; this prevents inadvertent overwriting of other flags in the register.

12.5.4 On-chip wakeups

The Wakeup Unit supports two on-chip wakeup sources. It combines the on-chip wakeups with the external ones to generate a single wakeup to the system.

On-chip wakeup management

In order to allow software to determine the wakeup source at one location, on-chip wakeups are reported along with external wakeups in the WISR register (see *Figure 98* for details). Enabling and clearing of these wakeups are done via the on-chip wakeup source's own registers.

13 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)

13.1 Overview

The RTC/API is a free running counter used for time keeping applications. The RTC may be configured to generate an interrupt at a predefined interval independent of the mode of operation (run mode or low power mode). If in a low power mode when the RTC interval is reached, the RTC first generates a wakeup and then assert the interrupt request. The RTC also supports an autonomous periodic interrupt (API) function used to generate a periodic wakeup request to exit a low power mode or an interrupt request.

13.2 Features

Features of the RTC/API include:

- 3 selectable counter clock sources
  - SIRC (128 kHz)
  - SXOSC (32 KHz)
  - FIRC (16 MHz)
- Optional 512 prescaler and optional 32 prescaler
- 32-bit counter
  - Supports times up to 1.5 months with 1 ms resolution
  - Runs in all modes of operation
  - Reset when disabled by software and by POR
- 12-bit compare value to support interrupt intervals of 1 s up to greater than 1 hr with 1 s resolution
- RTC compare value changeable while counter is running
- RTC status and control register are reset only by POR
- Autonomous periodic interrupt (API)
  - 10-bit compare value to support wakeup intervals of 1.0 ms to 1 s
  - Compare value changeable while counter is running
- Configurable interrupt for RTC match, API match, and RTC rollover
- Configurable wakeup event for RTC match, API match, and RTC rollover

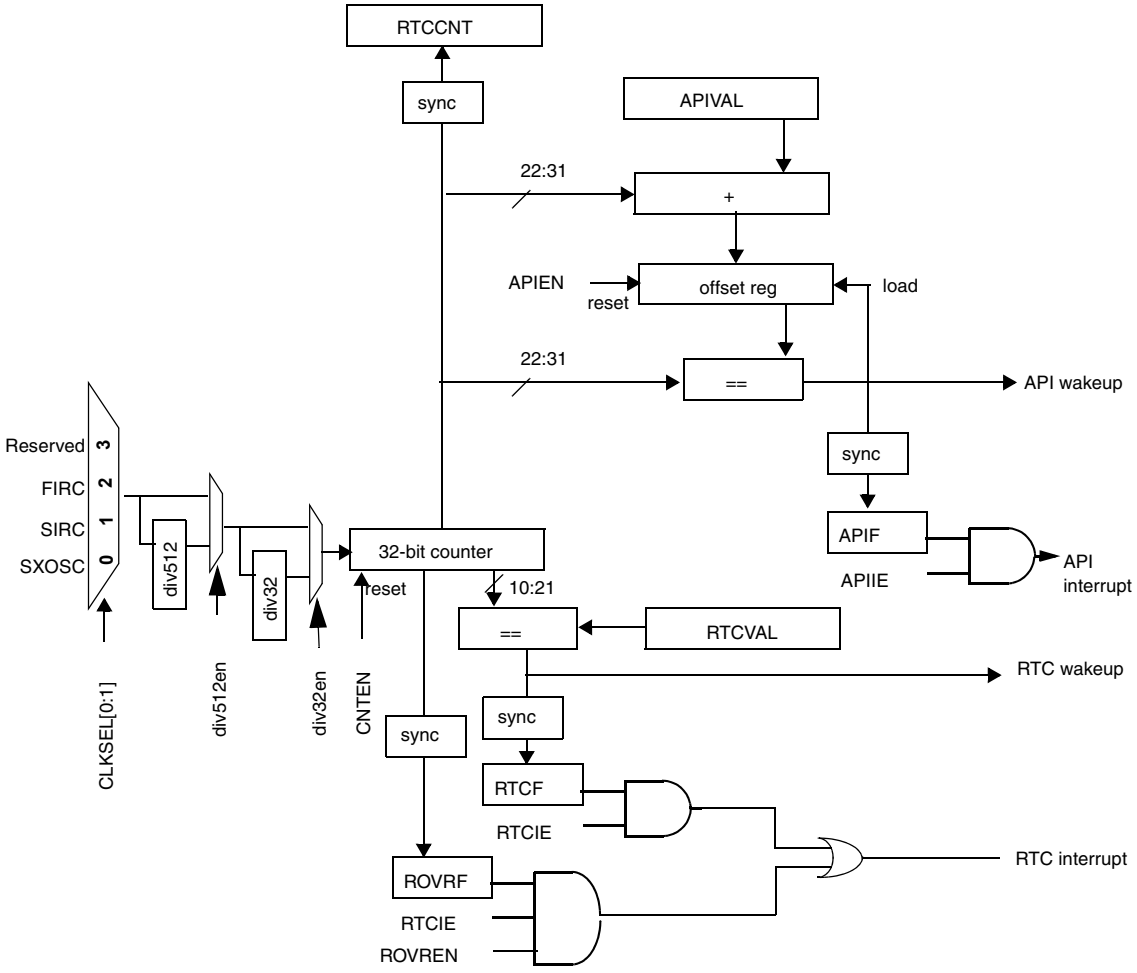


Figure 107. RTC/API block diagram

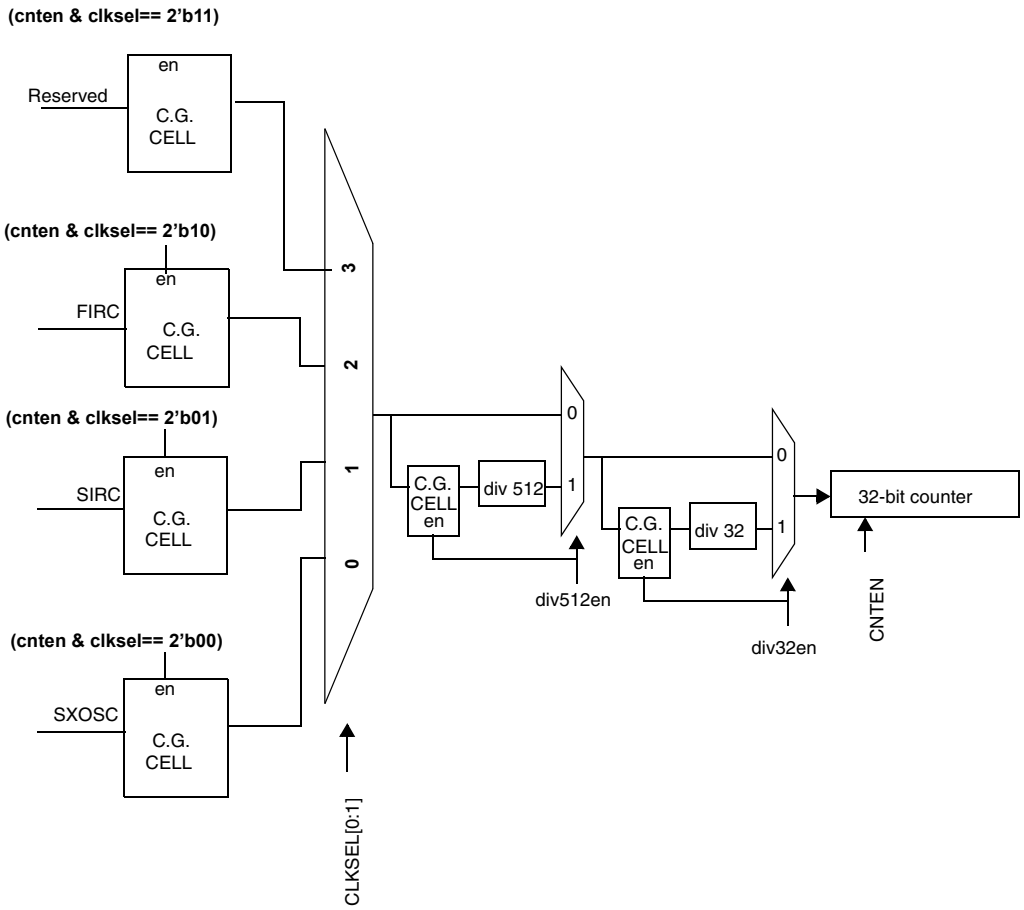


Figure 108. Clock gating for RTC clocks

13.3 Device-specific information

For SPC560Bx and SPC560Cx, the device specific information is the following:

- SXOSC, FIRC and SIRC clocks are provided as counter clocks for the RTC. Default clock on reset is SIRC divided by 4.
- The RTC will be reset on destructive reset, with the exception of software watchdog reset.
- The RTC provides a configurable divider by 512 to be optionally used when FIRC source is selected.

13.4 Modes of operation

13.4.1 Functional mode

There are two functional modes of operation for the RTC: normal operation and low power mode. In normal operation, all RTC registers can read or written and the input isolation is

disabled. The RTC/API and associated interrupts are optionally enabled. In low power mode, the bus interface is disabled and the input isolation is enabled. The RTC/API is enabled if enabled prior to entry into low power mode.

13.4.2 Debug mode

On entering into the debug mode the RTC counter freezes on the last valid count if the RTCC[FRZEN] is set. On exit from debug mode counter continues from the frozen value.

13.5 Register descriptions

Table 107 lists the RTC/API registers.

Table 107. RTC/API register map

Base address: 0xC3FE_C000		
Address offset	**Register**	**Location**
0x0	RTC Supervisor Control Register (RTCSUPV)	*on page 13-250*
0x4	RTC Control Register (RTCC)	*on page 13-251*
0x8	RTC Status Register (RTCS)	*on page 13-253*
0xC	RTC Counter Register (RTCCNT)	*on page 13-254*

13.5.1 RTC Supervisor Control Register (RTCSUPV)

The RTCSUPV register contains the SUPV bit which determines whether other registers are accessible in supervisor mode or user mode.

Note: RTCSUPV register is accessible only in supervisor mode.

Figure 109. RTC Supervisor Control Register (RTCSUPV)

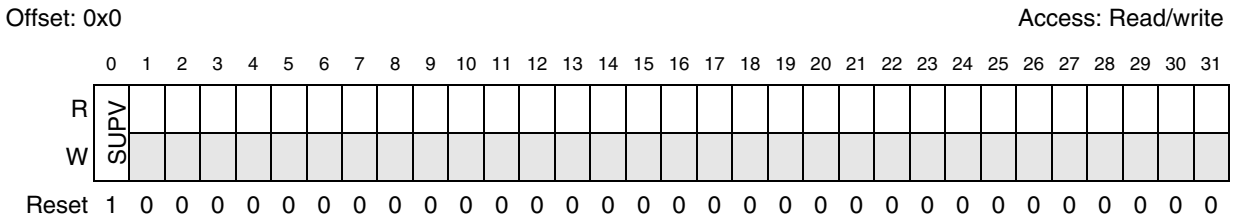


Table 108. RTCSUPV field descriptions

Field	Description
SUPV	RTC Supervisor Bit 0 All registers are accessible in both user as well as supervisor mode. 1 All other registers are accessible in supervisor mode only.

13.5.2 RTC Control Register (RTCC)

The RTCC register contains:

- RTC counter enable
- RTC interrupt enable
- RTC clock source select
- RTC compare value
- API enable
- API interrupt enable
- API compare value

Figure 110. RTC Control Register (RTCC)

Offset: 0x4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTEN	RTCIE	FRZEN	ROVREN	RTCVAL											
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	APIEN	APIIE	CLKSEL		DIV512EN	DIV32EN	APIVAL									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 109. RTCC field descriptions

Field	Description
CNTEN	Counter Enable The CNTEN field enables the RTC counter. Making CNTEN bit 1'b0 has the effect of asynchronously resetting (synchronous reset negation) all the RTC and API logic. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues. 1 Counter enabled 0 Counter disabled
RTCIE	RTC Interrupt Enable The RTCIE field enables interrupts requests to the system if RTCF is asserted. 1 RTC interrupts enabled 0 RTC interrupts disabled
FRZEN	Freeze Enable The counter freezes on entering the debug mode on the last valid count value if the FRZEN bit is set. After coming out of the debug mode, the counter starts from the frozen value. 0 Counter does not freeze in debug mode. 1 Counter freezes in debug mode.

Table 109. RTCC field descriptions (continued)

Field	Description
ROVREN	Counter Roll Over Wakeup/Interrupt Enable The ROVREN bit enables wakeup and interrupt requests when the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. The RTCIE bit must also be set in order to generate an interrupt from a counter rollover. 1 RTC rollover wakeup/interrupt enabled 0 RTC rollover wakeup/interrupt disabled
RTCVAL	RTC Compare Value The RTCVAL bits are compared to bits 10:21 of the RTC counter and if match sets RTCF. RTCVAL can be updated when the counter is running. *Note: RTCVAL = 0 does not generate an interrupt.*
APIEN	Autonomous Periodic Interrupt Enable The APIEN bit enables the autonomous periodic interrupt function. 1 API enabled 0 API disabled
APIIE	API Interrupt Enable The APIIE bit enables interrupts requests to the system if APIF is asserted. 1 API interrupts enabled 0 API interrupts disabled
CLKSEL	Clock Select This field selects the clock source for the RTC. CLKSEL may only be updated when CNTEN is 0. The user should ensure that oscillator is enabled before selecting it as a clock source for RTC. 00 SXOSC 01 SIRC 10 FIRC 11 Reserved
DIV512EN	Divide by 512 enable The DIV512EN bit enables the 512 clock divider. DIV512EN may only be updated when CNTEN is 0. 0 Divide by 512 is disabled. 1 Divide by 512 is enabled.
DIV32EN	Divide by 32 enable The DIV32EN bit enables the 32 clock divider. DIV32EN may only be updated when CNTEN is 0. 0 Divide by 32 is disabled. 1 Divide by 32 is enabled.
APIVAL	API Compare Value The APIVAL field is compared with bits 22:31 of the RTC counter and if match asserts an interrupt/wakeup request. APIVAL may only be updated when APIEN is 0 or API function is undefined. *Note: API functionality starts only when APIVAL is non zero. The first API interrupt takes two more cycles because of synchronization of APIVAL to the RTC clock. After that interrupts are periodic in nature. The minimum supported value of APIVAL is 4.*

13.5.3 RTC Status Register (RTCS)

The RTCS register contains:

- RTC interrupt flag
- API interrupt flag
- ROLLOVR Flag

Figure 111. RTC Status Register (RTCS)

Offset: 0x8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RTCF	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	APIF	0	0	ROVRF	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 110. RTCS field descriptions

Field	Description
RTCF	RTC Interrupt Flag The RTCF bit indicates that the RTC counter has reached the counter value matching RTCVAL. RTCF is cleared by writing a 1 to RTCF. Writing a 0 to RTCF has no effect. 1 RTC counter matches RTCVAL 0 RTC counter is not equal to RTCVAL
APIF	API Interrupt Flag The APIF bit indicates that the RTC counter has reached the counter value matching API offset value. APIF is cleared by writing a 1 to APIF. Writing a 0 to APIF has no effect. 1 API interrupt 0 No API interrupt Note: The periodic interrupt comes after APIVAL[0:9] + 1'b1 RTC counts
ROVRF	Counter Roll Over Interrupt Flag The ROVRF bit indicates that the RTC has rolled over from 0xffff_ffff to 0x0000_0000. ROVRF is cleared by writing a 1 to ROVRF. 1 RTC has rolled over 0 RTC has not rolled over

13.5.4 RTC Counter Register (RTCCNT)

The RTCCNT register contains the current value of the RTC counter.

Figure 112. RTC Counter Register (RTCCNT)

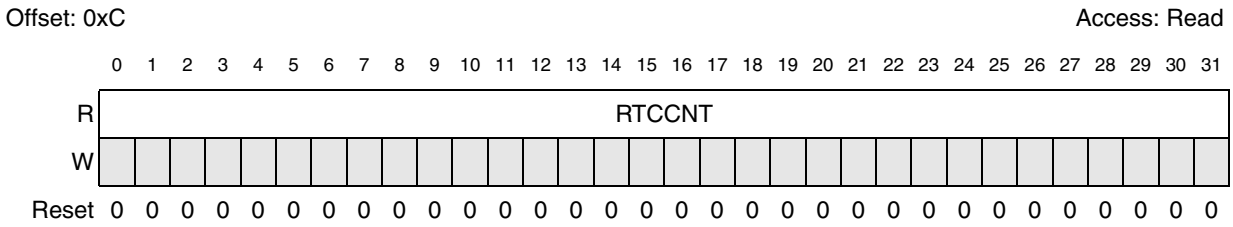


Table 111. RTCCNTfield descriptions

Field	Description
RTCCNT	RTC Counter Value Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value.

13.6 RTC functional description

The RTC consists of a 32-bit free running counter enabled with the RTCC[CNTEN] bit (CNTEN when negated asynchronously resets the counter and synchronously enables the counter when enabled). The value of the counter may be read via the RTCCNT register. Note that due to the clock synchronization, the RTCCNT value may actually represent a previous counter value. The difference between the counter and the read value depends on ratio of counter clock and system clock. Maximum possible difference between the two is 6 count values.

The clock source to the counter is selected with the RTCC[CLKSEL] field, which gives the options for clocking the RTC/API. The output of the clock mux can be optionally divided by combination of 512 and 32 to give a 1 ms RTC/API count period for different clock sources. Note that the RTCC[CNTEN] bit must be disabled when the RTC/API clock source is switched.

When the counter value for counter bits 10:21 match the 12-bit value in the RTCC[RTCVAL] field, then the RTCS[RTCF] interrupt flag bit is set (after proper clock synchronization). If the RTCC[RTCIE] interrupt enable bit is set, then the RTC interrupt request is generated. The RTC supports interrupt requests in the range of 1 s to 4096 s (> 1 hr) with a 1 s resolution. If there is a match while in low power mode then the RTC will first generate a wakeup request to force a wakeup to run mode, then the RTCF flag will be set.

A rollover wakeup and/or interrupt can be generated when the RTC transitions from a count of 0xFFFF_FFFF to 0x0000_0000. The rollover flag is enabled by setting the RTCC[ROVREN] bit. An RTC counter rollover with this bit will cause a wakeup from low power mode. An interrupt request is generated for an RTC counter rollover when both the RTCC[ROVREN] and RTCC[RTCIE] bits are set.

All the flags and counter values are synchronized with the system clock. It is assumed that the system clock frequency is always more than or equal to the rtc_clk used to run the counter.

13.7 API functional description

Setting the RTCC[APIEN] bit enables the autonomous interrupt function. The 10-bit RTCC[APIVAL] field selects the time interval for triggering an interrupt and/or wakeup event. Since the RTC is a free running counter, the APIVAL is added to the current count to calculate an offset. When the counter reaches the offset count, a interrupt and/or wakeup request is generated. Then the offset value is recalculated and again re-triggers a new request when the new value is reached. APIVAL may only be updated when APIEN is disabled. When a compare is reached, the RTCS[APIF] interrupt flag bit is set (after proper clock synchronization). If the RTCC[APIIE] interrupt enable bit is set, then the API interrupt request is generated. If there is a match while in low power mode, then the API will first generate a wakeup request to force a wakeup into normal operation, then the APIF flag will be set.

14 CAN Sampler

14.1 Introduction

The CAN sampler peripheral has been designed to store the first identifier of CAN message “detected” on the CAN bus while no precise clock (crystal) is running at that time on the device, typically in low power modes (STOP, HALT or STANDBY) or in RUN mode with crystal switched off.

Depending on both CAN baud rate and low power mode used, it is possible to catch either the first or the second CAN frame by sampling one CAN Rx port among six and storing all samples in internal registers.

After selection of the mode (first or second frame), the CAN sampler stores samples of the 48 bits or skips the first frame and stores samples of the 48 bits of second frame using the 16 MHz fast internal RC oscillator and the 5-bit clock prescaler.

After completion, software has to process the sampled data in order to rebuild the 48 minimal bits.

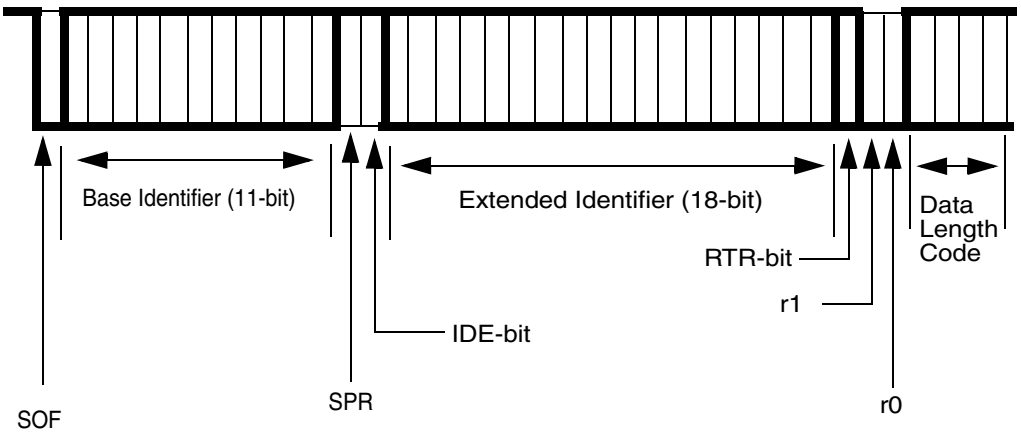


Figure 113. Extended CAN data frame

14.2 Main features

- Store 384 samples, equivalent to 48 CAN bit @ 8 samples/bit
- Sample frequency from 500 kHz up to 16 MHz, equivalent at 8 samples/bit to CAN baud rates of 62.5 Kbps to 2 Mbps
- User selectable CAN Rx sample port [CAN0RX-CAN5RX]
- 16 MHz fast internal RC oscillator clock
- 5-bit clock prescaler
- Configurable trigger mode (immediate, next frame)
- Flexible samples processing by software
- Very low power consumption

14.3 Register description

The CAN sampler registers are listed in *Table 112*.

Table 112. CAN sampler memory map

Base address: 0xFFE7_0000		
Address offset	**Register**	**Location**
0x00	Control Register (CR)	*on page 14-257*
0x04–0x30	Sample registers 0–11	*on page 14-258*

14.3.1 Control Register (CR)

Offset: 0x00 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	RX_COMPLETE	BUSY	ACTIVE_CK	0	0	0	MODE	CAN_RX_SEL			BRP					CAN_SMPLR_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 114. Control Register (CR)

Table 113. CR field descriptions

Field	Description
RX_COMPLETE	1: CAN frame is stored in the sample registers 0: CAN frame has not been stored in the sample registers
BUSY	This bit indicates the sampling status 1: Sampling is ongoing 0: Sampling is complete or has not started
ACTIVE_CK	This bit indicates which is current clock for sample registers, that is, xmem_ck. 1: RC_CLK is currently xmem_ck 0: ipg_clk_s is currently xmem_ck
MODE	0: Skip the first frame and sample and store the second frame (SF_MODE) 1: Sample and store the first frame (FF_MODE)

Table 113. CR field descriptions (continued)

Field	Description
CAN_RX_SEL	This field determines which RX port is sampled. One Rx port can be selected per sampling routine. 000: CAN0RX PB[1] is selected 001: CAN1RX PC[11] is selected 010: CAN2RX PE[9] is selected 011: CAN3RX PE[9] is selected 100: CAN4RX PC[11] is selected 101: CAN5RX PE[0] is selected 110: Reserved 111: Reserved
BRP	Baud Rate Prescaler This field is used to set the baud rate before going into STANDBY mode. 00000: Prescaler has 1 11111: Prescaler has 32
CAN_SMPLR_EN	CAN Sampler Enable This bit enables the CAN sampler before going into STANDBY or STOP mode. 0: CAN sampler is disabled 1: CAN sampler is enabled

14.3.2 Sample register n (n = 0..11)

Offsets: 0x04–0x30 (12 registers) Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SR[31:16]															
Reset	The reset values are unknown. They will be filled only after the first CAN sampling.															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	SR[15:0]															
Reset	The reset values are unknown. They will be filled only after the first CAN sampling.															

Figure 115. Sample register n

14.4 Functional description

As the CAN sampler is driven by the 16 MHz fast internal RC oscillator (or “FIRC”) to properly sample the CAN identifier, two modes are possible depending on both the CAN baud rate and low power mode used:

- Immediate sampling on falling edge detection (first CAN frame): This mode is used when the FIRC is available in LP mode (for example, STOP or HALT).
- Sampling on next frame (second CAN frame): This mode is used when the FIRC is switched off in LP mode (for example, STANDBY). Due to the start-up times of both the voltage regulator and the FIRC (~10 µs), the CAN sampler would miss the first bits of a CAN identifier sent at 500 kbps. Therefore, the first identifier is ignored and the sampling is performed on the first falling edge of after interframe space.

The CAN sampler is in power domain 0 and maintains register settings throughout low power modes. The CAN sampler performs sampling on a user-selected CAN Rx port among six Rx ports available, normally when the device is in STANDBY or STOP mode, storing the samples in internal registers. The user is required to configure the baud rate to achieve eight samples per CAN nominal bit. It does not perform any sort of filtering on input samples. Thereafter the software must enable the sampler by setting the CAN_SMPLR_EN bit in the CR register. It then becomes the master controller for accessing the internal registers implemented for storing samples.

The CAN sampler, when enabled, waits for a low pulse on the selected Rx line, taking it as a valid bit of the first CAN frame and generates the RC wakeup request which can be used to start the FIRC. Depending upon the mode, it stores the first 8 samples of the 48 bits on selected Rx line or skips the first frame and stores 8 bits for first 48 bits of second frame. In FF_MODE, it samples the CAN Rx line on the FIRC clock and stores the 8 samples of first 48 bits (384 samples). In SF_MODE, it samples the Rx and waits for 11 consecutive dominant bits (11 * 8 samples), taking it as the end of first frame. It then waits for first low pulse on the Rx, taking it as a valid Start of Frame (SOF) of the second frame. The sampler takes 384 samples (48 bytes * 8) using the FIRC clock (configuring 8 samples per nominal bit) of the second frame, including the SOF bit. These samples are stored in consecutive addresses of the (12 x 32) internal registers. The RX_COMPLETE bit is set to ‘1’, indicating that sampling is complete.

Software should now process the sampled data by first becoming master for accessing samples internal registers by resetting the CAN_SMPLR_EN bit. The sampler will need to be enabled again to start waiting for a new sampling routine.

14.4.1 Enabling/Disabling the CAN sampler

The CAN sampler is disabled on reset and the CPU is able to access the 12 registers used for storing samples. The CAN sampler must be enabled before going into STANDBY or STOP mode by setting the CAN_SMPLR_EN bit in the Control Register (CR) by writing ‘1’ to this bit.

In case of any activity on the selected Rx line, the sampler enables the 16 MHz fast internal RC oscillator. When bit CAN_SMPLR_EN is reset to 0, the sampler should receive at last three FIRC clock pulses to reset itself, after which the FIRC can be switched off.

When the software attempts to access the sample registers’ contents it must first reset the CAN_SMPLR_EN bit by writing a ‘0’. Before accessing the register contents it must monitor Active_CK bit for ‘0’. When this bit is reset it can safely access the (12 x 32) sample

registers. While shifting from normal to sample mode and from sample to normal mode, the sample register signals must be static and inactive to ensure the data is not corrupt.

14.4.2 Baud rate generation

Sampling is performed at a baud rate that is set by the software as a multiple of RC oscillator frequency of 62.5 ns (assuming RC is configured for high frequency mode, that is, 16 MHz). The user must set the baud rate prescaler (BRP) such that eight samples per bit are achieved.

The baud rate setting must be made by software before going into STANDBY or STOP mode. This is done by setting bits BRP[4:0] in the Control register. The reset value of BRP is 00000 and can be set to max. 11111 which gives a prescale value of BRP + 1, thus providing a BRP range of 1 to 32.

- Maximum bitrate supported for sampling is 2 Mbps using BRP as 1
- Minimum bitrate supported for sampling is 62.5 kbps using BRP as 32

For example, suppose the system is transmitting at 125 kbps. In this case, nominal bit period:

Equation 2 $T=1/(125*10^{3})s = 8*10^{-3}*10^{-3}s = 8\mu s$

To achieve 8 samples per bit

Sample period= 8/8 µs = 1 µs

BRP = 1 µs/62.5 ns = 16. Thus in this case BRP = 01111

15 e200z0h Core

15.1 Overview

The e200 processor family is a set of CPU cores that implement cost-efficient versions of the Power Architecture®. e200 processors are designed for deeply embedded control applications which require low cost solutions rather than maximum performance.

The 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 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 technology support, the e200z0h core only implements the VLE (variable-length encoding) APU, providing improved code density.

15.2 Microarchitecture summary

The e200z0h 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 8x32 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 e200z0h. Prefetched instructions are placed into an instruction buffer with 4entries in e200z0h, each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches which 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 platform. 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.

15.3 Block diagram

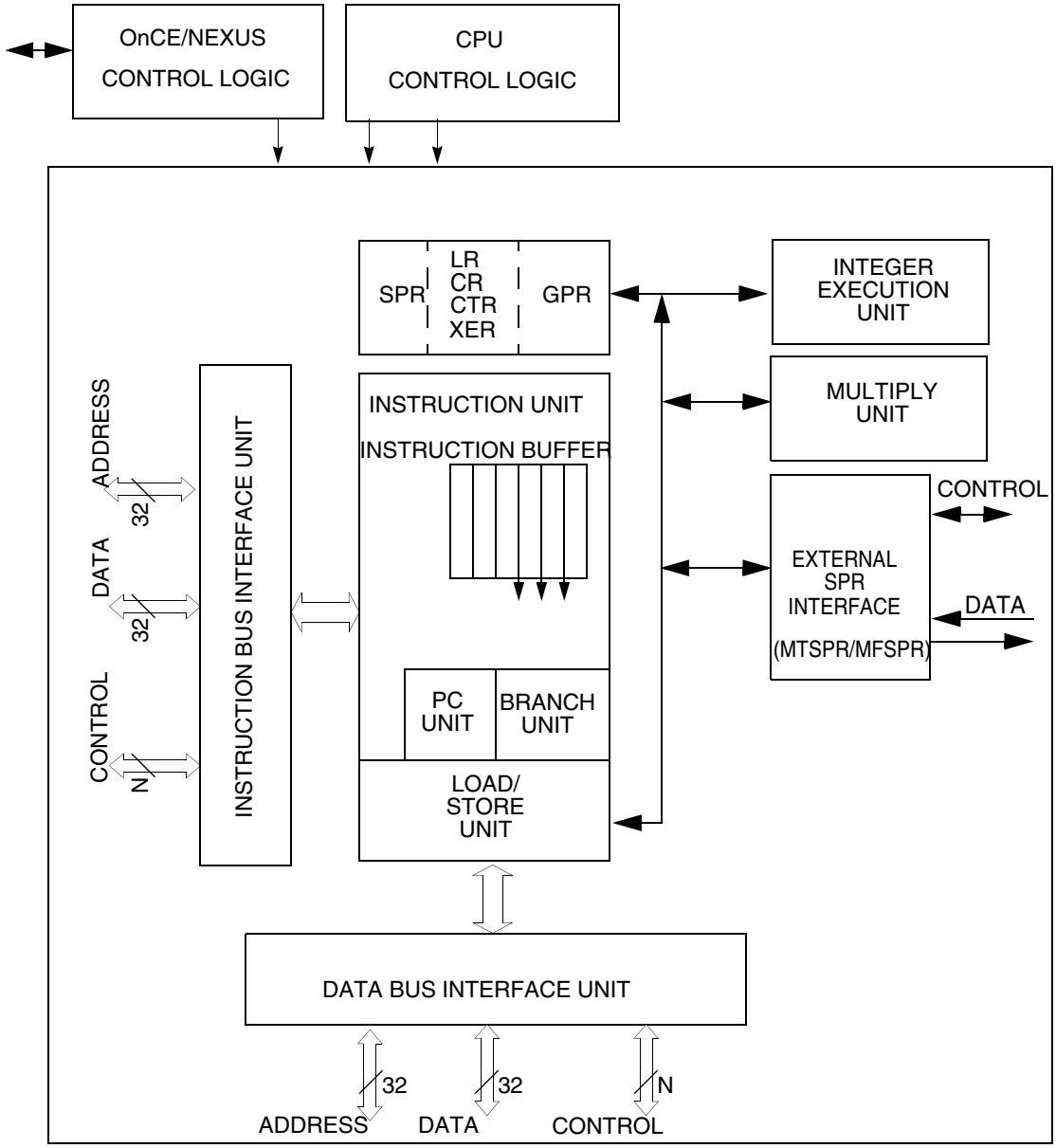


Figure 116. e200z0h block diagram

15.4 Features

The following is a list of some of the key features of the e200z0h core:

- 32-bit Power Architecture 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
- 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).
- 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
  - Power saving modes: nap, sleep, and wait
  - Dynamic power management of execution units
- Testability
  - Synthesizeable, full MuxD scan design
  - ABIST/MBIST for optional memory arrays

15.4.1 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 up to 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
- 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

15.4.2 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
- 8x32 hardware multiplier array supports 1 to 4 cycle 32x32->32 multiply (early out)

15.4.3 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)

15.4.4 e200z0h system bus features

The features of the e200z0h system bus interface are as follows:

- Independent instruction and data buses
- AMBA[l] AHB[m] 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

15.4.5 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.

l. Advanced Microcontroller Bus Architecture

m. Advanced High Performance Bus

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

15.5 Core registers and programmer's model

This section describes the registers implemented in the e200z0h cores. It includes an overview of registers defined by the Power Architecture platform, 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 the 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 117, and *Figure 116* show the e200 register set including the registers which are accessible while in supervisor mode, and the registers which 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: e200z0h is a 32-bit implementation of the Power Architecture specification. In this document, register bits are sometimes numbered from bit 0 (Most Significant Bit) to 31 (Least Significant Bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher.

Where appropriate, the Book E defined bit numbers are shown in parentheses.

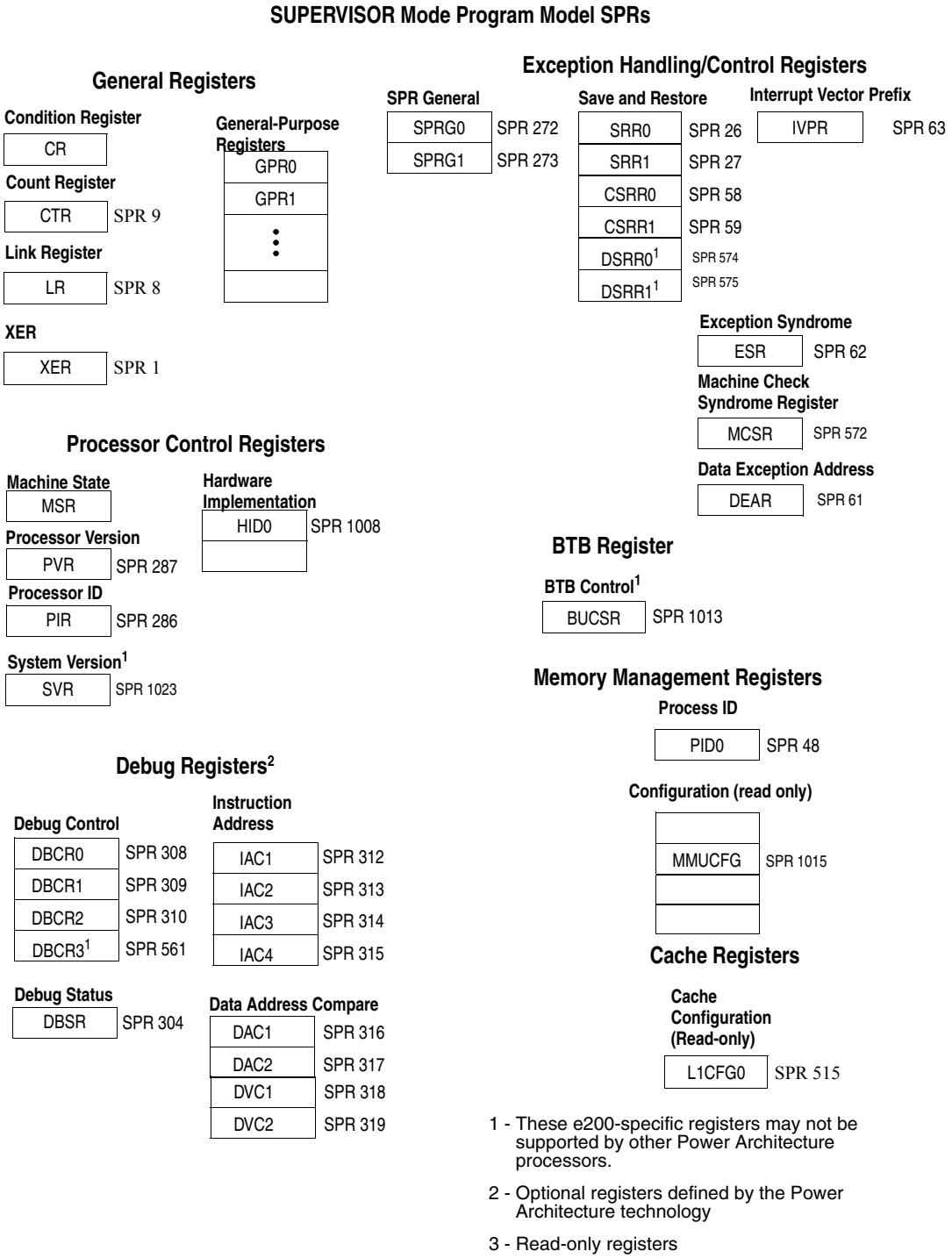


Figure 117. e200z0 SUPERVISOR Mode Program Model SPRs

16 Interrupt Controller (INTC)

16.1 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 142 interrupt requests. It is targeted to work with a Power Architecture technology processor and automotive powertrain applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.

For high priority interrupt requests in these target applications, the time from the assertion of the peripheral's interrupt request from the peripheral 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. Since 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 all tasks which share the resource cannot preempt each other.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests. These same software configurable interrupt requests also can be used to break 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.

16.2 Features

- Supports 134 peripheral and 8 software-configurable interrupt request sources
- Unique 9-bit vector per interrupt source
- Each interrupt source can be programmed to one of 16 priorities
- 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 clocks from receipt of interrupt request from peripheral to interrupt request to processor

Table 114. Interrupt sources available

Interrupt sources (142)	Number available
Software	8
ECSM	1
Software Watchdog (SWT)	1
STM	4
Flash/SRAM ECC (SEC-DED)	2
Real Time Counter (RTC/API)	2
System Integration Unit Lite (SIUL)	2
WKPU	3
MC_ME	4
MC_RGM	1
FXOSC	1
PIT	6
ADC_0	3
FlexCAN_0	8
FlexCAN_1	8
FlexCAN_2	8
FlexCAN_3	8
FlexCAN_4	8
FlexCAN_5	8
LINFlex_0	3
LINFlex_1	3
LINFlex_2	3
LINFlex_3	3
DSPI_0	5
DSPI_1	5
DSPI_2	5
I2C_0	1
Enhanced Modular I/O Subsystem 0 (eMIOS_0)	14
eMIOS_1	14

16.3 Block diagram

Figure 118 provides a block diagram of the INTC.

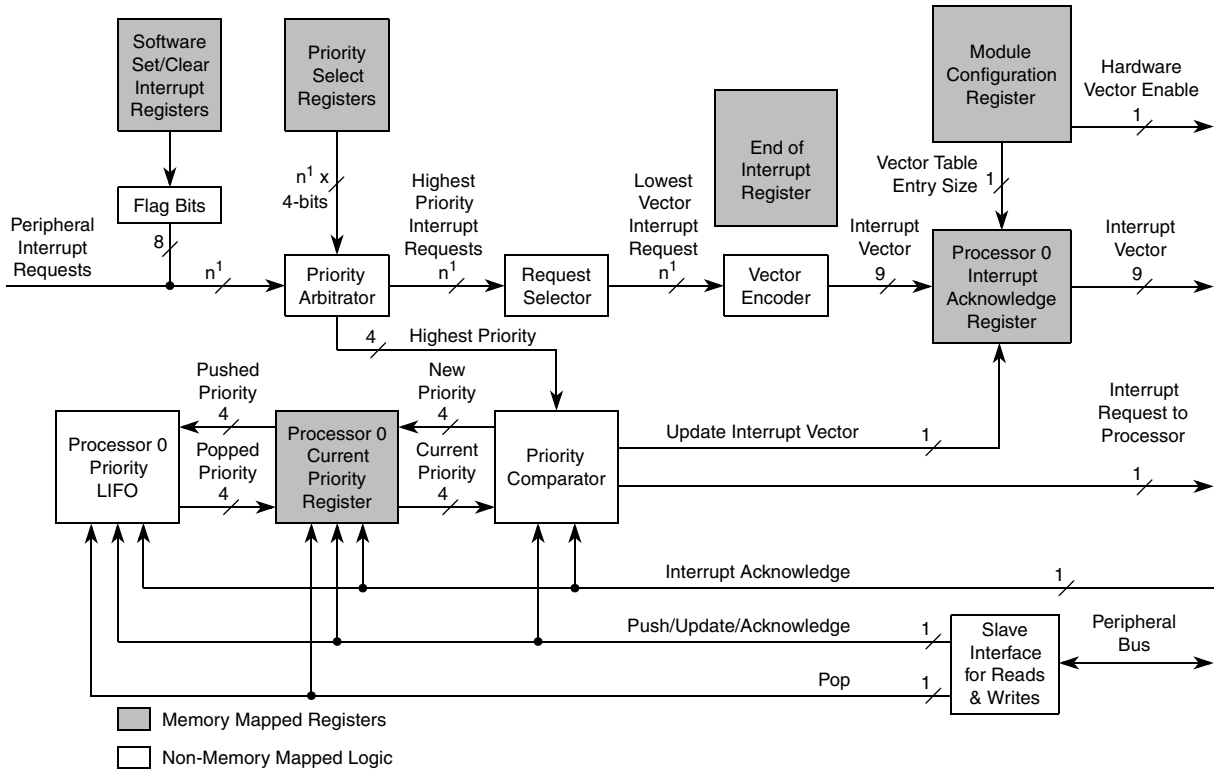


Figure 118. INTC block diagram

16.4 Modes of operation

16.4.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

Software vector mode

In software vector mode, software, that is the interrupt exception handler, 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 is the interrupt vector signal from the INTC in conjunction with a processor with the capability use that vector. In hardware vector mode, 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. Since the INTC is not clocked in STOP mode, peripheral interrupt requests can not be used as a wakeup source, unless the device supports that interrupt request as a wakeup source.

16.5 Memory map and register description

16.5.1 Module memory map

Table 115 shows the INTC memory map.

Table 115. INTC memory map

Base address: 0xFFF4_8000		
Address offset	**Register**	**Location**
0x0000	INTC Module Configuration Register (INTC_MCR)	*on page 16-272*
0x0004	Reserved	
0x0008	INTC Current Priority Register for Processor (INTC_CPR)	*on page 16-273*
0x000C	Reserved	
0x0010	INTC Interrupt Acknowledge Register (INTC_IACKR)	*on page 16-275*
0x0014	Reserved	
0x0018	INTC End-of-Interrupt Register (INTC_EOIR)	*on page 16-276*
0x001C	Reserved	
0x0020–0x0027	INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)	*on page 16-276*
0x0028–0x003C	Reserved	
0x0040–0x00D0	INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR208_210)(1)	*on page 16-278*

1. The PRI fields are “reserved” for peripheral interrupt requests whose vectors are labeled ‘Reserved’ in *Figure 120*.

16.5.2 Register 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 eight bits, aligned 16 bits, misaligned 16 bits to the middle two 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 Module Configuration Register (INTC_MCR)

The module configuration register is used to configure options of the INTC.

Figure 119. INTC Module Configuration Register (INTC_MCR)

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	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 116. INTC_MCR field descriptions

Field	Description
VTES	Vector table entry size. Controls the number of '0's 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 vectortable as in software vector mode, then the number of rightmost '0's 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
HVEN	Hardware vector enable. Controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 16.4 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 for Processor (INTC_CPR)

Figure 120. INTC Current Priority Register (INTC_CPR)

Offset: 0x0008 Access: User read/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	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	PRI			
W																																
Reset	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	1	1	1	1

Table 117. INTC_CPR field descriptions

Field	Description
PRI	Priority PRI is the priority of the currently executing ISR according to the field values defined in *Table 118*.

The INTC_CPR masks any peripheral or software configurable 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 16.7.5 Priority ceiling protocol*.

Note: A store to modify the PRI field which closely precedes or follows an access to a shared resource can result in a non-coherent access to that resource. Refer to Section Ensuring coherency for example code to ensure coherency.

Table 118. PRI values

PRI	Meaning
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

INTC Interrupt Acknowledge Register (INTC_IACKR)

Figure 121. INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 0

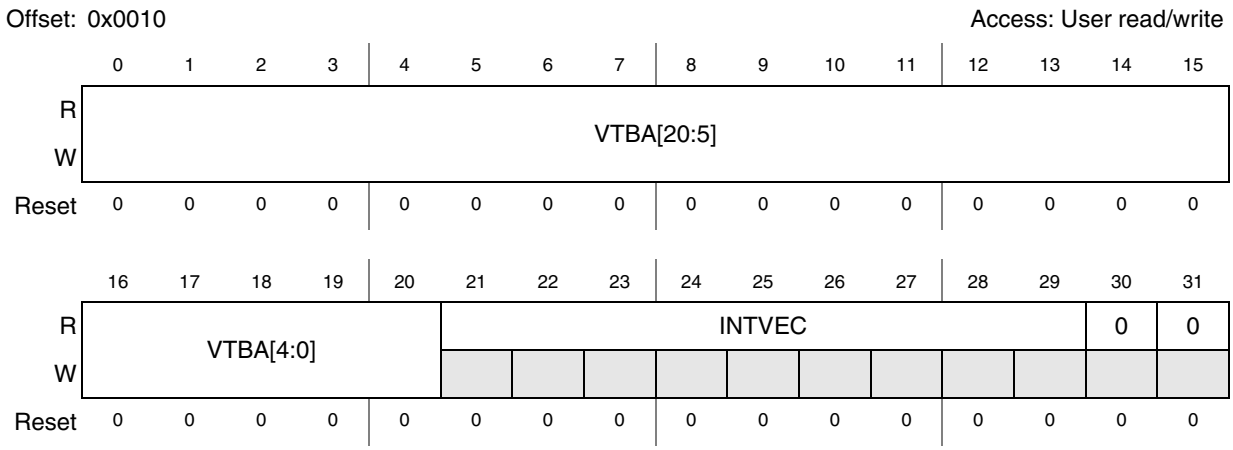


Figure 122. INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 1

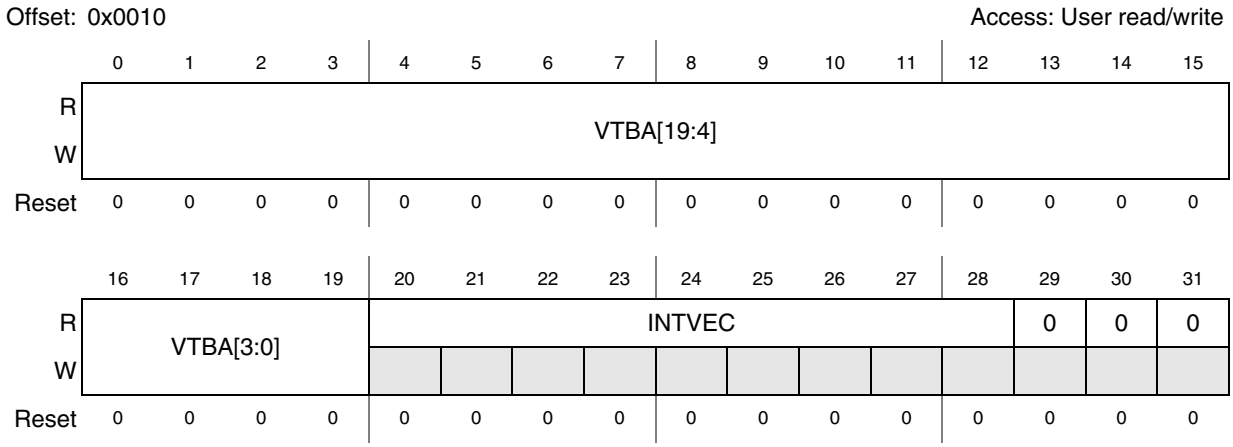


Table 119. INTC_IACKR field descriptions

Field	Description
VTBA	Vector Table Base Address Can be the base address of a vector table of addresses of ISRs.
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.

The interrupt acknowledge register provides a value which 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 123. INTC End-of-Interrupt Register (INTC_EOIR)

Offset: 0x0018 Access: Write 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	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
W	See text																															
Reset	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

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–INTC_SSCIR4_7)

Figure 124. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

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	CLR0	0	0	0	0	0	0	0	CLR1
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	CLR2	0	0	0	0	0	0	0	CLR3
W							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 125. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

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	CLR4	0	0	0	0	0	0	0	CLR5
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	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 120. INTC_SSCIR[0:7] field descriptions

Field	Description
SETx	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.
CLRx	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_PSR208_210)

Figure 126. INTC Priority Select Register 0–3 (INTC_PSR[0:3])

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	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 127. INTC Priority Select Register 208-210 (INTC_PSR[208:210])

Offset: 0x0110 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	PRI208				0	0	0	0	PRI209			
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	PRI210				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 121. INTC_PSR0_3–INTC_PSR208_210 field descriptions

Field	Description
PRI	Priority Select PRI*x* selects the priority for interrupt requests. See *Section 16.6 Functional description*.

Table 122. INTC Priority Select Register address offsets

INTC_PSR*x*_*x*	Offset address	INTC_PSR*x*_*x*	Offset address
INTC_PSR0_3	0x0040	INTC_PSR108_111	0x00AC
INTC_PSR4_7	0x0044	INTC_PSR112_115	0x00B0
INTC_PSR8_11	0x0048	INTC_PSR116_119	0x00B4
INTC_PSR12_15	0x004C	INTC_PSR120_123	0x00B8
INTC_PSR16_19	0x0050	INTC_PSR124_127	0x00BC

Table 122. INTC Priority Select Register address offsets (continued)

INTC_PSR*x*_*x*	Offset address	INTC_PSR*x*_*x*	Offset address
INTC_PSR20_23	0x0054	INTC_PSR128_131	0x00C0
INTC_PSR24_27	0x0058	INTC_PSR132_135	0x00C4
INTC_PSR28_31	0x005C	INTC_PSR136_139	0x00C8
INTC_PSR32_35	0x0060	INTC_PSR140_143	0x00CC
INTC_PSR36_39	0x0064	INTC_PSR144_147	0x00D0
INTC_PSR40_43	0x0068	INTC_PSR148_151	0x00D4
INTC_PSR44_47	0x006C	INTC_PSR152_155	0x00D8
INTC_PSR48_51	0x0070	INTC_PSR156_159	0x00DC
INTC_PSR52_55	0x0074	INTC_PSR160_163	0x00E0
INTC_PSR56_59	0x0078	INTC_PSR164_167	0x00E4
INTC_PSR60_63	0x007C	INTC_PSR168_171	0x00E8
INTC_PSR64_67	0x0080	INTC_PSR172_175	0x00EC
INTC_PSR68_71	0x0084	INTC_PSR176_179	0x00F0
INTC_PSR72_75	0x0088	INTC_PSR180_183	0x00F4
INTC_PSR76_79	0x008C	INTC_PSR184_187	0x00F8
INTC_PSR80_83	0x0090	INTC_PSR188_191	0x00FC
INTC_PSR84_87	0x0094	INTC_PSR192_195	0x0100
INTC_PSR88_91	0x0098	INTC_PSR196_199	0x0104
INTC_PSR92_95	0x009C	INTC_PSR200_203	0x0108
INTC_PSR96_99	0x00A0	INTC_PSR204_207	0x010C
INTC_PSR100_103	0x00A4	INTC_PSR208_210	0x0110
INTC_PSR104_107	0x00A8	—	—

16.6 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_PSR210 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 123. Interrupt vector table

IRQ #	Offset	Size (bytes)	Interrupt	Module
Section A (Core Section)				
—	0x0000	16	Critical Input (INTC software vector mode) / NMI	Core
—	0x0010	16	Machine check / NMI	Core
—	0x0020	16	Data Storage	Core
—	0x0030	16	Instruction Storage	Core
—	0x0040	16	External Input (INTC software vector mode)	Core
—	0x0050	16	Alignment	Core
—	0x0060	16	Program	Core
—	0x0070	16	Reserved	Core
—	0x0080	16	System call	Core
—	0x0090	96	Unused	Core
—	0x00F0	16	Debug	Core
—	0x0100	1792	Unused	Core
Section B (On-Platform Peripherals)				
0	0x0800	4	Software configurable flag 0	Software
1	0x0804	4	Software configurable flag 1	Software
2	0x0808	4	Software configurable flag 2	Software
3	0x080C	4	Software configurable flag 3	Software
4	0x0810	4	Software configurable flag 4	Software
5	0x0814	4	Software configurable flag 5	Software
6	0x0818	4	Software configurable flag 6	Software
7	0x081C	4	Software configurable flag 7	Software
8	0x0820	4	Reserved	
9	0x0824	4	Platform Flash Bank 0 Abort | Platform Flash Bank 0 Stall | Platform Flash Bank 1 Abort | Platform Flash Bank 1 Stall |	ECSM
10	0x0828	4	Reserved	
11	0x082C	4	Reserved	
12	0x0830	4	Reserved	
13	0x0834	4	Reserved	
14	0x0838	4	Reserved	
15	0x083C	4	Reserved	

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
16	0x0840	4	Reserved	
17	0x0844	4	Reserved	
18	0x0848	4	Reserved	
19	0x084C	4	Reserved	
20	0x0850	4	Reserved	
21	0x0854	4	Reserved	
22	0x0858	4	Reserved	
23	0x085C	4	Reserved	
24	0x0860	4	Reserved	
25	0x0864	4	Reserved	
26	0x0868	4	Reserved	
27	0x086C	4	Reserved	
28	0x0870	4	Timeout	SWT
29	0x0874	4	Reserved	
30	0x0878	4	Match on channel 0	STM
31	0x087C	4	Match on channel 1	STM
32	0x0880	4	Match on channel 2	STM
33	0x0884	4	Match on channel 3	STM
34	0x0888	4	Reserved	
35	0x088C	4	ECC_DBD_PlatformFlash | ECC_DBD_PlatformRAM	Platform ECC Double Bit Detection
36	0x0890	4	ECC_SBC_PlatformFlash | ECC_SBC_PlatformRAM	Platform ECC Single Bit Correction
37	0x0894	4	Reserved	
Section C				
38	0x0898	4	RTC	RTC/API
39	0x089C	4	API	RTC/API
40	0x08A0	4	Reserved	
41	0x08A4	4	SIU External IRQ_0	SIUL
42	0x08A8	4	SIU External IRQ_1	SIUL
43	0x08AC	4	Reserved	
44	0x08B0	4	Reserved	
45	0x08B4	4	Reserved	
46	0x08B8	4	WakeUp_IRQ_0	WKPU
47	0x08BC	4	WakeUp_IRQ_1	WKPU

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
48	0x08C0	4	WakeUp_IRQ_2	WKPU
49	0x08C4	4	Reserved	
50	0x08C8	4	Reserved	
51	0x08CC	4	Safe Mode Interrupt	MC_ME
52	0x08D0	4	Mode Transition Interrupt	MC_ME
53	0x08D4	4	Invalid Mode Interrupt	MC_ME
54	0x08D8	4	Invalid Mode Config	MC_ME
55	0x08DC	4	Reserved	
56	0x08E0	4	Functional and destructive reset alternate event interrupt (ipi_int)	MC_RGM
57	0x08E4	4	FXOSC counter expired (ipi_int_osc)	FXOSC
58	0x08E8	4	Reserved	
59	0x08EC	4	PITimer Channel 0	PIT
60	0x08F0	4	PITimer Channel 1	PIT
61	0x08F4	4	PITimer Channel 2	PIT
62	0x08F8	4	ADC_EOC	ADC_0
63	0x08FC	4	ADC_ER	ADC_0
64	0x0900	4	ADC_WD	ADC_0
65	0x0904	4	FlexCAN_ESR[ERR_INT]	FlexCAN_0
66	0x0908	4	FlexCAN_ESR_BOFF | FlexCAN_Transmit_Warning | FlexCAN_Receive_Warning	FlexCAN_0
67	0x090C	4	Reserved	
68	0x0910	4	FlexCAN_BUF_00_03	FlexCAN_0
69	0x0914	4	FlexCAN_BUF_04_07	FlexCAN_0
70	0x0918	4	FlexCAN_BUF_08_11	FlexCAN_0
71	0x091C	4	FlexCAN_BUF_12_15	FlexCAN_0
72	0x0920	4	FlexCAN_BUF_16_31	FlexCAN_0
73	0x0924	4	FlexCAN_BUF_32_63	FlexCAN_0
74	0x0928	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_0
75	0x092C	4	DSPI_SR[EOQF]	DSPI_0
76	0x0930	4	DSPI_SR[TFFF]	DSPI_0
77	0x0934	4	DSPI_SR[TCF]	DSPI_0
78	0x0938	4	DSPI_SR[RFDF]	DSPI_0

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
79	0x093C	4	LINFlex_RXI	LINFlex_0
80	0x0940	4	LINFlex_TXI	LINFlex_0
81	0x0944	4	LINFlex_ERR	LINFlex_0
82	0x0948	4	Reserved	
83	0x094C	4	Reserved	
84	0x0950	4	Reserved	
85	0x0954	4	FlexCAN_ESR[ERR_INT]	FlexCAN_1
86	0x0958	4	FlexCAN_ESR_BOFF | FlexCAN_Transmit_Warning | FlexCAN_Receive_Warning	FlexCAN_1
87	0x095C	4	Reserved	
88	0x0960	4	FlexCAN_BUF_00_03	FlexCAN_1
89	0x0964	4	FlexCAN_BUF_04_07	FlexCAN_1
90	0x0968	4	FlexCAN_BUF_08_11	FlexCAN_1
91	0x096C	4	FlexCAN_BUF_12_15	FlexCAN_1
92	0x0970	4	FlexCAN_BUF_16_31	FlexCAN_1
93	0x0974	4	FlexCAN_BUF_32_63	FlexCAN_1
94	0x0978	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_1
95	0x097C	4	DSPI_SR[EOQF]	DSPI_1
96	0x0980	4	DSPI_SR[TFFF]	DSPI_1
97	0x0984	4	DSPI_SR[TCF]	DSPI_1
98	0x0988	4	DSPI_SR[RFDF]	DSPI_1
99	0x098C	4	LINFlex_RXI	LINFlex_1
100	0x0990	4	LINFlex_TXI	LINFlex_1
101	0x0994	4	LINFlex_ERR	LINFlex_1
102	0x0998	4	Reserved	
103	0x099C	4	Reserved	
104	0x09A0	4	Reserved	
105	0x09A4	4	FlexCAN_[ERR_INT]	FlexCAN_2
106	0x09A8	4	FlexCAN_ESR_BOFF | FlexCAN_Transmit_Warning | FlexCAN_Receive_Warning	FlexCAN_2
107	0x09AC	4	Reserved	
108	0x09B0	4	FlexCAN_BUF_00_03	FlexCAN_2

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
109	0x09B4	4	FlexCAN_BUF_04_07	FlexCAN_2
110	0x09B8	4	FlexCAN_BUF_08_11	FlexCAN_2
111	0x09BC	4	FlexCAN_BUF_12_15	FlexCAN_2
112	0x09C0	4	FlexCAN_BUF_16_31	FlexCAN_2
113	0x09C4	4	FlexCAN_BUF_32_63	FlexCAN_2
114	0x09C8	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_2
115	0x09CC	4	DSPI_SR[EOQF]	DSPI_2
116	0x09D0	4	DSPI_SR[TFFF]	DSPI_2
117	0x09D4	4	DSPI_SR[TCF]	DSPI_2
118	0x09D8	4	DSPI_SR[RFDF]	DSPI_2
119	0x09DC	4	LINFlex_RXI	LINFlex_2
120	0x09E0	4	LINFlex_TXI	LINFlex_2
121	0x09E4	4	LINFlex_ERR	LINFlex_2
122	0x09E8	4	LINFlex_RXI	LINFlex_3
123	0x09EC	4	LINFlex_TXI	LINFlex_3
124	0x09F0	4	LINFlex_ERR	LINFlex_3
125	0x09F4	4	I2C_SR[IBAL] I2C_SR[TCF] I2C_SR[IAAS]	I2C_0
126	0x09F8	4	Reserved	
127	0x09FC	4	PITimer Channel 3	PIT
128	0x0A00	4	PITimer Channel 4	PIT
129	0x0A04	4	PITimer Channel 5	PIT
130	0x0A08	4	Reserved	
131	0x0A0C	4	Reserved	
132	0x0A10	4	Reserved	
133	0x0A14	4	Reserved	
134	0x0A18	4	Reserved	
135	0x0A1C	4	Reserved	
136	0x0A20	4	Reserved	
137	0x0A24	4	Reserved	
138	0x0A28	4	Reserved	
139	0x0A2C	4	Reserved	
140	0x0A30	4	Reserved	

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
141	0x0A34	4	EMIOS_GFR[F0,F1]	eMIOS_0
142	0x0A38	4	EMIOS_GFR[F2,F3]	eMIOS_0
143	0x0A3C	4	EMIOS_GFR[F4,F5]	eMIOS_0
144	0x0A40	4	EMIOS_GFR[F6,F7]	eMIOS_0
145	0x0A44	4	EMIOS_GFR[F8,F9]	eMIOS_0
146	0x0A48	4	EMIOS_GFR[F10,F11]	eMIOS_0
147	0x0A4C	4	EMIOS_GFR[F12,F13]	eMIOS_0
148	0x0A50	4	EMIOS_GFR[F14,F15]	eMIOS_0
149	0x0A54	4	EMIOS_GFR[F16,F17]	eMIOS_0
150	0x0A58	4	EMIOS_GFR[F18,F19]	eMIOS_0
151	0x0A5C	4	EMIOS_GFR[F20,F21]	eMIOS_0
152	0x0A60	4	EMIOS_GFR[F22,F23]	eMIOS_0
153	0x0A64	4	EMIOS_GFR[F24,F25]	eMIOS_0
154	0x0A68	4	EMIOS_GFR[F26,F27]	eMIOS_0
155	0x0A6C	4	Reserved	
156	0x0A70	4	Reserved	
Section D (Device specific vectors)				
157	0x0A74	4	EMIOS_GFR[F0,F1]	eMIOS_1
158	0x0A78	4	EMIOS_GFR[F2,F3]	eMIOS_1
159	0x0A7C	4	EMIOS_GFR[F4,F5]	eMIOS_1
160	0x0A80	4	EMIOS_GFR[F6,F7]	eMIOS_1
161	0x0A84	4	EMIOS_GFR[F8,F9]	eMIOS_1
162	0x0A88	4	EMIOS_GFR[F10,F11]	eMIOS_1
163	0x0A8C	4	EMIOS_GFR[F12,F13]	eMIOS_1
164	0x0A90	4	EMIOS_GFR[F14,F15]	eMIOS_1
165	0x0A94	4	EMIOS_GFR[F16,F17]	eMIOS_1
166	0x0A98	4	EMIOS_GFR[F18,F19]	eMIOS_1
167	0x0A9C	4	EMIOS_GFR[F20,F21]	eMIOS_1
168	0x0AA0	4	EMIOS_GFR[F22,F23]	eMIOS_1
169	0x0AA4	4	EMIOS_GFR[F24,F25]	eMIOS_1
170	0x0AA8	4	EMIOS_GFR[F26,F27]	eMIOS_1
171	0x0AAC	4	Reserved	
172	0x0AB0	4	Reserved	
173	0x0AB4	4	FlexCAN_ESR	FlexCAN_3

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
174	0x0AB8	4	FlexCAN_ESR_BOFF | FlexCAN_Transmit_Warning | FlexCAN_Receive_Warning	FlexCAN_3
175	0x0ABC	4	Reserved	
176	0x0AC0	4	FlexCAN_BUF_0_3	FlexCAN_3
177	0x0AC4	4	FlexCAN_BUF_4_7	FlexCAN_3
178	0x0AC8	4	FlexCAN_BUF_8_11	FlexCAN_3
179	0x0ACC	4	FlexCAN_BUF_12_15	FlexCAN_3
180	0x0AD0	4	FlexCAN_BUF_16_31	FlexCAN_3
181	0x0AD4	4	FlexCAN_BUF_32_63	FlexCAN_3
182	0x0AD8	4	Reserved	
183	0x0ADC	4	Reserved	
184	0x0AE0	4	Reserved	
185	0x0AE4	4	Reserved	
186	0x0AE8	4	Reserved	
187	0x0AEC	4	Reserved	
188	0x0AF0	4	Reserved	
189	0x0AF4	4	Reserved	
190	0x0AF8	4	FlexCAN_ESR	FlexCAN_4
191	0x0AFC	4	FlexCAN_ESR_BOFF | FlexCAN_Transmit_Warning | FlexCAN_Receive_Warning	FlexCAN_4
192	0x0B00	4	Reserved	
193	0x0B04	4	FlexCAN_BUF_0_3	FlexCAN_4
194	0x0B08	4	FlexCAN_BUF_4_7	FlexCAN_4
195	0x0B0C	4	FlexCAN_BUF_8_11	FlexCAN_4
196	0x0B10	4	FlexCAN_BUF_12_15	FlexCAN_4
197	0x0B14	4	FlexCAN_BUF_16_31	FlexCAN_4
198	0x0B18	4	FlexCAN_BUF_32_63	FlexCAN_4
199	0x0B1C	4	Reserved	
200	0x0B20	4	Reserved	
201	0x0B24	4	Reserved	
202	0x0B28	4	FlexCAN_ESR	FlexCAN_5

Table 123. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
203	0x0B2C	4	FlexCAN_ESR_BOFF | FlexCAN_Transmit_Warning | FlexCAN_Receive_Warning	FlexCAN_5
204	0x0B30	4	Reserved	
205	0x0B34	4	FlexCAN_BUF_0_3	FlexCAN_5
206	0x0B38	4	FlexCAN_BUF_4_7	FlexCAN_5
207	0x0B3C	4	FlexCAN_BUF_8_11	FlexCAN_5
208	0x0B40	4	FlexCAN_BUF_12_15	FlexCAN_5
209	0x0B44	4	FlexCAN_BUF_16_31	FlexCAN_5
210	0x0B48	4	FlexCAN_BUF_32_63	FlexCAN_5
211	0x0B4C	4	Reserved	
212	0x0B50	4	Reserved	
213	0x0B54	4	Reserved	
214	0x0B58	4	Reserved	
215	0x0B5C	4	Reserved	
216	0x0B60	4	Reserved	

16.6.1 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 19.6.3 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 114*).

16.6.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*x* values set in the INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR208_210). 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 118* 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 configurable 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 which 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, the only the one with the lowest vector is passed 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 subblock compares the highest priority output from the priority arbitrator subblock with PRI in INTC_CPR. If the priority comparator subblock detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the associated 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 which 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 softwarevector 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 ‘0’s 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.

16.6.3 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, along with the handshaking near the end of the interrupt exception handler, is shown in *Figure 128*. 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 the handshaking 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 Power Architecture® 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.

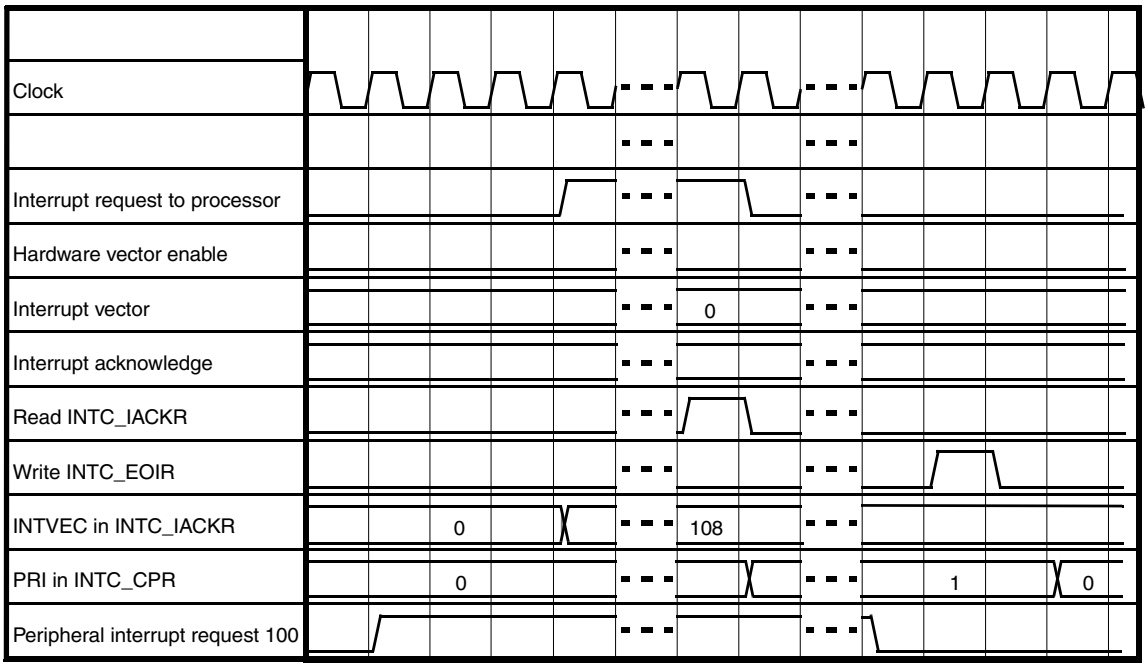


Figure 128. Software vector mode handshaking timing diagram

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 129*. 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*.

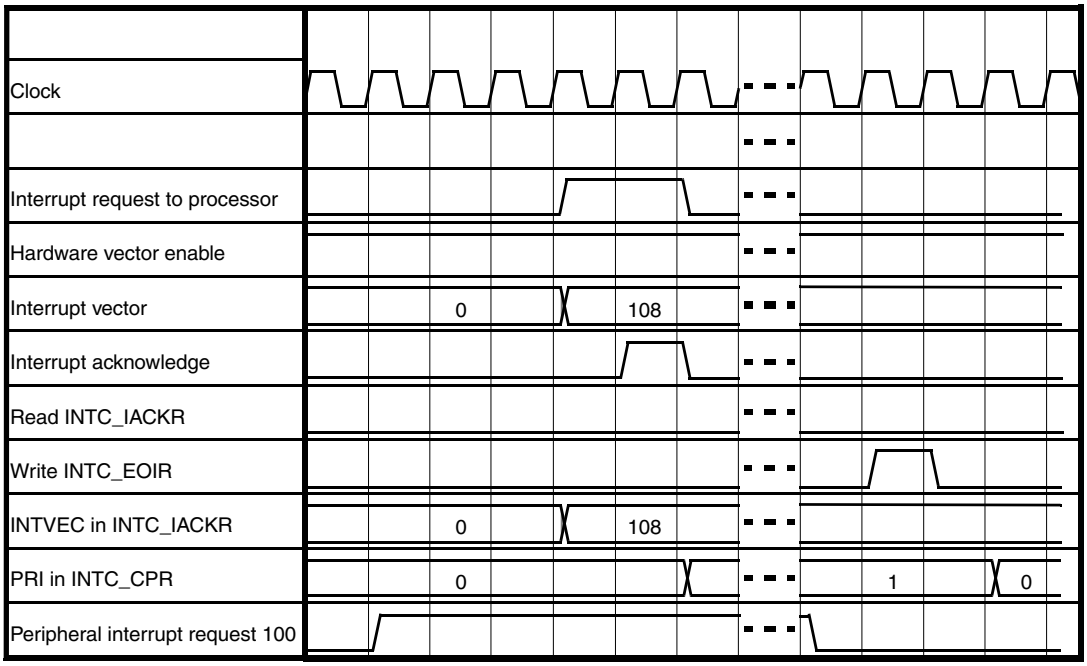


Figure 129. Hardware vector mode handshaking timing diagram

16.7 Initialization/application information

16.7.1 Initialization flow

After exiting reset, all of the PRI*n* fields in INTC priority select registers (INTC_PSR0–INTC_PSR210) 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
```

16.7.2 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 which drives interrupt request to INTC

blr # return to epilog
```

Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations which 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 which drives interrupt request to INTC
blr             # branch to epilog
```

16.7.3 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–INTC_PSR210) 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.

16.7.4 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 124* shows the order of execution of both ISRs with different priorities and the same priority.

Table 124. Order of ISR execution example

Step No.	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
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1

Table 124. Order of ISR execution example (continued)

Step No.	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	
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.

16.7.5 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
```

16.7.6 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 which 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.

16.7.7 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 the INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR208_210), 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 which 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.

16.7.8 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 the INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR208_210) 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.

16.7.9 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 *the* INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR208_210) 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*.

16.7.10 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
```

17 Crossbar Switch (XBAR)

17.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between two master ports and three slave ports. XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

17.2 Block diagram

Figure 130 shows a block diagram of the crossbar switch.

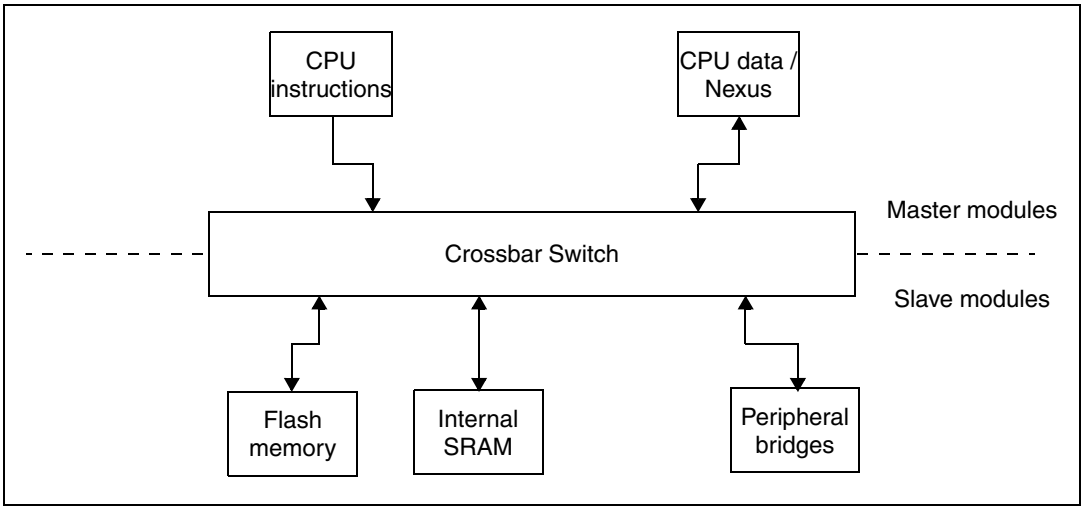


Figure 130. XBAR block diagram

Table 125 gives the crossbar switch port for each master and slave, and the assigned and fixed ID number for each master. The table shows the master ID numbers as they relate to the master port numbers.

Table 125. XBAR switch ports for SPC560Bx and SPC560Cx

Module	Port		Physical master ID
	Type	Logical number	
e200z0 core–CPU instructions	Master	0	0
e200z0 core–CPU data / Nexus	Master	0	1
Flash memory	Slave	0	—
Internal SRAM	Slave	2	—
Peripheral bridges	Slave	7	—

17.3 Overview

The XBAR allows 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 selects the higher priority master and grants 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 granted access based on a fixed priority.

17.4 Features

- 2 master ports:
  - Core: e200z0 core instructions
  - Core: e200z0 core data / Nexus
- 3 slave ports
  - Flash (refer to the flash memory chapter for information on accessing flash memory)
  - Internal SRAM
  - Peripheral bridges
- 32-bit address, 32-bit data paths
- Fully concurrent transfers between independent master and slave ports
- Fixed priority scheme and fixed parking strategy

17.5 Modes of operation

17.5.1 Normal mode

In normal mode, the XBAR provides the logic that controls crossbar switch configuration.

17.5.2 Debug mode

The XBAR operation in debug mode is identical to operation in normal mode.

17.6 Functional description

This section describes the functionality of the XBAR in more detail.

17.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. 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 stalls masters, or inserts bubbles on the slave side.

17.6.2 General operation

When a master makes an access to the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the XBAR appears to be simply another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a 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 the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed.

When a slave bus is idled by the XBAR, it is parked on the master which did the last transfer.

17.6.3 Master ports

A master access is taken if 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 is completely transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master.

If the slave port is currently parked on another master, and no other master is requesting access to the slave port, then only one clock of arbitration is 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 gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly responds 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.

17.6.4 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. 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 forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master which does not own the slave port is granted access after a one clock delay.

17.6.5 Priority assignment

Each master port is assigned a fixed 3-bit priority level (hard-wired priority). The following table shows the priority levels assigned to each master (the lowest has highest priority).

Table 126. Hardwired bus master priorities

Module	Port		Priority level
	Type	Number	
e200z0 core–CPU instructions	Master	0	7
e200z0 core–CPU data / Nexus	Master	0	6

17.6.6 Arbitration

XBAR supports only a fixed-priority comparison algorithm.

Fixed priority operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR_MPR. If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains 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 (if any). 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 higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But 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 is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks always to the last master (park-on-last). When parked on the last master, the slave port is passing that master's signals through to the slave bus. When the master accesses

the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.

18 Memory Protection Unit (MPU)

18.1 Introduction

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in the device. Using preprogrammed 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 module provides the following capabilities:

- Support for 8 program-visible 128-bit (4-word) region descriptors
  - Each region descriptor defines a modulo-32 byte space, aligned anywhere in memory

    Region sizes can vary from a minimum of 32 bytes to a maximum of 4 Gbytes
  - Two types of access control permissions defined in single descriptor word

    Processors have separate {read, write, execute} attributes for supervisor and user accesses

    Non-processor masters have {read, write} attributes
  - Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
  - Alternate programming model view of the access control permissions word
- Memory-mapped platform device
  - Interface to 3 slave XBAR ports: flash controller, system SRAM controller and peripherals bus

    Connections to the address phase address and attributes

    Typical location is immediately “downstream” of the platform’s crossbar switch

A simplified block diagram of the MPU module is shown in *Figure 131*.

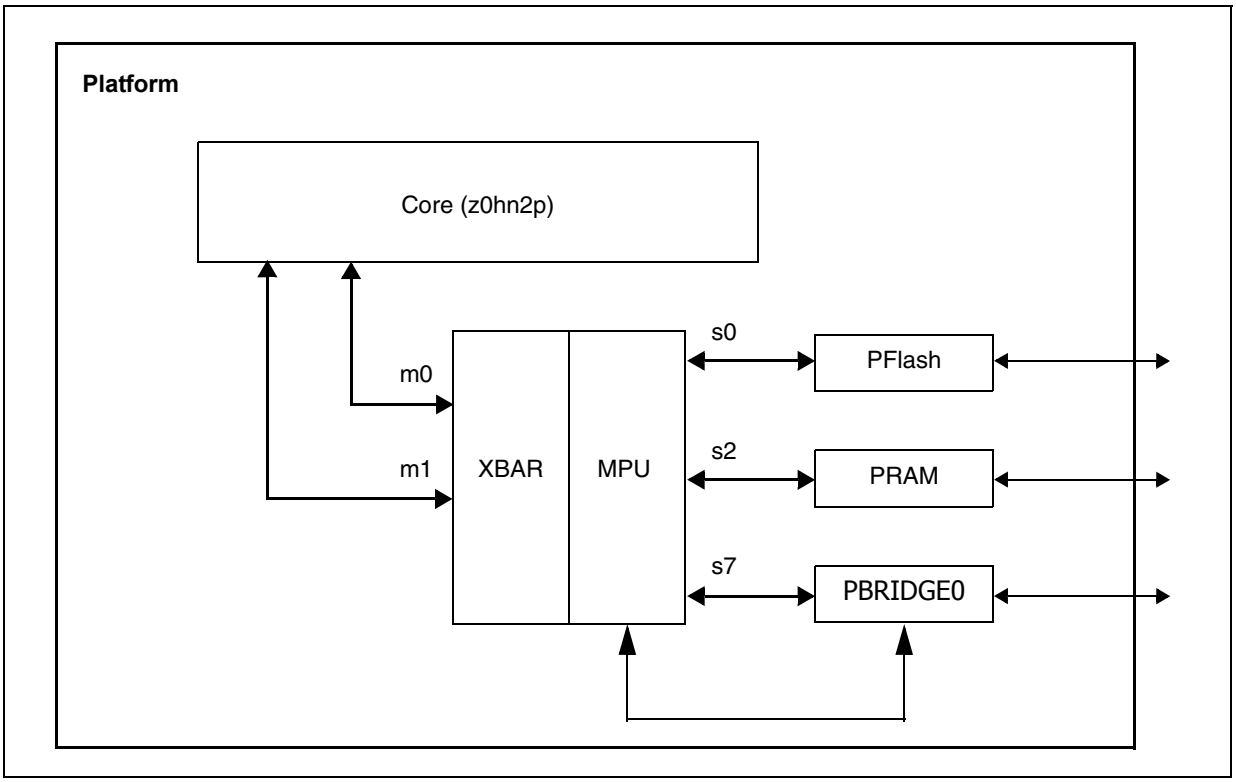


Figure 131. MPU block diagram

18.2 Features

The Memory Protection Unit implements a two-dimensional hardware array of memory region descriptors and the crossbar slave XBAR ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for 8 memory region descriptors, each 128 bits in size
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - Access control definitions: 2 bus masters (processor cores) 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 18.6.2 Putting it all together and AHB error terminations for details and *Section 18.8 Application information* for an example.

- Support for 3 XBAR slave port connections: flash controller, system SRAM controller and peripherals bus
  - MPU hardware continuously monitors every XBAR slave port access using the preprogrammed 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 XBAR reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device.
  - 64-bit error registers, one for each XBAR 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

18.3 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 peripheral 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 XBAR system bus port(s).

Power dissipation is minimized when the MPU’s global enable/disable bit is cleared (MPU_CESR[VLD] = 0).

18.4 External signal description

The MPU module does not include any external interface. The MPU’s internal interfaces include a peripheral bus connection for accessing the programming model and multiple connections to the address phase signals of the platform crossbar’s slave AHB ports. From a platform topology viewpoint, the MPU module appears to be directly connected “downstream” from the crossbar switch with interfaces to the XBAR slave ports.

18.5 Memory map and register description

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 and 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.

Finally, the programming model allocates space for an MPU definition with 8 region descriptors and up to 3 XBAR slave ports, like flash controller, system SRAM controller and peripheral bus.

18.5.1 Memory map

The MPU programming model map is shown in *Table 127*.

Table 127. MPU memory map

Base address: 0xFFF1_1000		
Address offset	**Register**	**Location**
0x000	MPU Control/Error Status Register (MPU_CESR)	*on page 18-308*
0x004–0x00F	Reserved	
0x010	MPU Error Address Register, Slave Port 0 (MPU_EAR0)	*on page 18-309*
0x014	MPU Error Detail Register, Slave Port 0 (MPU_EDR0)	*on page 18-309*
0x018	MPU Error Address Register, Slave Port 1 (MPU_EAR1)	*on page 18-309*
0x01C	MPU Error Detail Register, Slave Port 1 (MPU_EDR1)	*on page 18-309*
0x020	MPU Error Address Register, Slave Port 2 (MPU_EAR2)	*on page 18-309*
0x024	MPU Error Detail Register, Slave Port 2 (MPU_EDR2)	*on page 18-309*
0x028–0x3FF	Reserved	
0x400	MPU Region Descriptor 0 (MPU_RGD0)	*on page 18-311*
0x410	MPU Region Descriptor 1 (MPU_RGD1)	*on page 18-311*
0x420	MPU Region Descriptor 2 (MPU_RGD2)	*on page 18-311*
0x430	MPU Region Descriptor 3 (MPU_RGD3)	*on page 18-311*
0x440	MPU Region Descriptor 4 (MPU_RGD4)	*on page 18-311*
0x450	MPU Region Descriptor 5 (MPU_RGD5)	*on page 18-311*
0x460	MPU Region Descriptor 6 (MPU_RGD6)	*on page 18-311*
0x470	MPU Region Descriptor 7 (MPU_RGD7)	*on page 18-311*
0x480–0x7FF	Reserved	
0x800	MPU RGD Alternate Access Control 0 (MPU_RGDAAC0)	*on page 18-316*
0x804	MPU RGD Alternate Access Control 1 (MPU_RGDAAC1)	*on page 18-316*
0x808	MPU RGD Alternate Access Control 2 (MPU_RGDAAC2)	*on page 18-316*
0x80C	MPU RGD Alternate Access Control 3 (MPU_RGDAAC3)	*on page 18-316*
0x810	MPU RGD Alternate Access Control 4 (MPU_RGDAAC4)	*on page 18-316*
0x814	MPU RGD Alternate Access Control 5 (MPU_RGDAAC5)	*on page 18-316*
0x818	MPU RGD Alternate Access Control 6 (MPU_RGDAAC6)	*on page 18-316*
0x81C	MPU RGD Alternate Access Control 7 (MPU_RGDAAC7)	*on page 18-316*

18.5.2 Register description

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.

Figure 132. MPU Control/Error Status Register (MPU_CESR)

Offset: 0x000 Access: Read/Partial Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SPERR[7:0]								1	0	0	0	HRL			
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c								
Reset	0	0	0	0	0	0	0	0	1	0	0	0	*	*	*	*

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

Table 128. MPU_CESR field descriptions

Field	Description
SPERRn	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 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 field specifies the number of slave ports [1–8] connected to the MPU.
NRGD	Number of Region Descriptors This field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0b0000 8 region descriptors 0b0001 12 region descriptors 0b0010 16 region descriptors

Table 128. MPU_CESR field descriptions (continued)

Field	Description
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.

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. Note 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 133. MPU Error Address Register, Slave Port n (MPU_EARn)

Offsets: 0x010–0x020 (3 registers) Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EADDR [31:16]															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 129. MPU_EARn field descriptions

Field	Description
EADDR	Error Address This field is the reference address from slave port n that generated the access error.

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 that 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 134. MPU Error Detail Register, Slave Port n (MPU_EDRn)

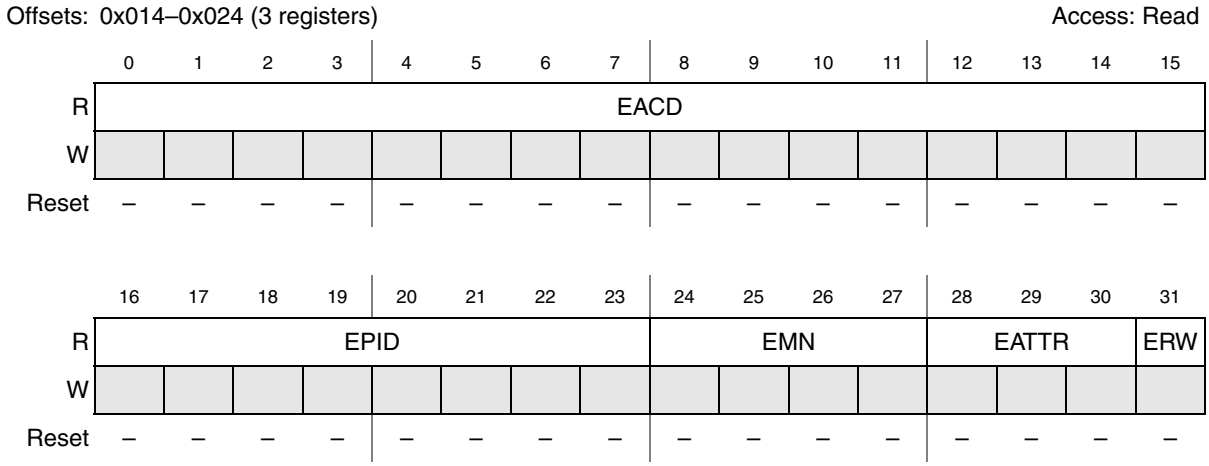


Table 130. MPU_EDRn field descriptions

Field	Description
EACD	Error Access Control Detail This field implements one bit per region descriptor and is an indication of the region descriptor hit logically ANDed with the access error indication. 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.
EPID	Error Process Identification This 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.
EMN	Error Master Number This field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
EATTR	Error Attributes This field records attribute information about the faulting reference. The supported encodings are defined as: 0b000 User mode, instruction access 0b001 User mode, data access 0b010Supervisor mode, instruction access 0b011Supervisor mode, data access All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).

Table 130. MPU_EDRn field descriptions (continued)

Field	Description
ERW	Error Read/Write This field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

MPU Region Descriptor n (MPU_RGDn)

Each 128-bit (16 byte) 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 135. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0)

Offset: 0x400 + (16*n) + 0x0 (MPU_RGDn.Word0) Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SRTADDR[26:11]															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SRTADDR[10:0]											0	0	0	0	0
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	0	0	0	0	0

Table 131. MPU_RGDn.Word0 field descriptions

Field	Description
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 136. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1)

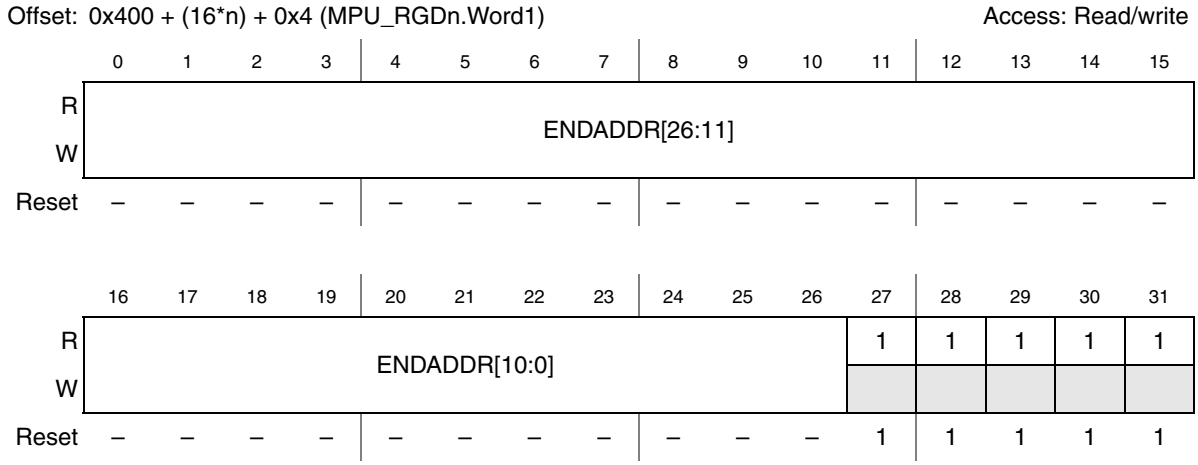


Table 132. MPU_RGDn.Word1 field descriptions

Field	Description
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. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are typically reserved for processor cores and the corresponding access control is 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. Bus masters 4–7 are typically reserved for data movement engines and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the logical master number defined as the XBAR hmaster[3:0] signal.

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.

The evaluation logic defines the processor access type based on multiple AHB signals, as hwrite and hprot[1:0].

For non-processor data movement engines (bus masters 4–7), the evaluation logic simply uses hwrite to determine if the access is a read or write.

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.

Figure 137. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2)

Offset: 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
R W	M7RE	M7WE	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	M3PE	M3SM			M3UM		M2PE	M2SM[1]
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	M2SM[0]	M2UM			M1PE	M1SM			M1UM			M0PE	M0SM		M0UM	
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 133. MPU_RGDn.Word2 field descriptions

Field	Description
M7RE	Bus master 7 read enable If set, this flag allows bus master 7 to perform read operations. If cleared, any attempted read by bus master 7 terminates with an access error and the read is not performed.
M7WE	Bus master 7 write enable If set, this flag allows bus master 7 to perform write operations. If cleared, any attempted write by bus master 7 terminates with an access error and the write is not performed.
M6RE	Bus master 6 read enable If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.
M6WE	Bus master 6 write enable If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.
M5RE	Bus master 5 read enable If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.
M5WE	Bus master 5 write enable If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.
M4RE	Bus master 4 read enable If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.

Table 133. MPU_RGDn.Word2 field descriptions (continued)

Field	Description
M4WE	Bus master 4 write enable If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.
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 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 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 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 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 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

Table 133. MPU_RGDn.Word2 field descriptions (continued)

Field	Description
M1UM	Bus master 1 user mode access control This 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 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 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 128-bit 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 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.

Figure 138. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3)

Offset: 0x400 + (16*n) + 0xC (MPU_RGDn.Word3) Access: Read/write

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

	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	VLD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 134. MPU_RGDn.Word3 field descriptions

Field	Description
PID	Process Identifier This 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.
PIDMASK	Process Identifier Mask This 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*.
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

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.

Figure 139. MPU RGD Alternate Access Control n (MPU_RGDAACn)

Offset: 0x800 + (4*n) (MPU_RGDAACn) Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	M7RE	M7WE	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	M3PE	M3SM		M3UM			M2PE	M2SM[1]
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	M2SM[0]	M2UM			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Since the MPU_RGDAACn register is simply another memory mapping for MPU_RGDn.Word2, the field definitions shown in *Table 135* are identical to those presented in *Table 133*.

Table 135. MPU_RGDAACn field descriptions

Field	Description
M7RE	Bus master 7 read enable. If set, this flag allows bus master 7 to perform read operations. If cleared, any attempted read by bus master 7 terminates with an access error and the read is not performed.
M7WE	Bus master 7 write enable If set, this flag allows bus master 7 to perform write operations. If cleared, any attempted write by bus master 7 terminates with an access error and the write is not performed.
M6RE	Bus master 6 read enable If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.
M6WE	Bus master 6 write enable If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.
M5RE	Bus master 5 read enable If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.
M5WE	Bus master 5 write enable If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.
M4RE	Bus master 4 read enable If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.
M4WE	Bus master 4 write enable If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.

Table 135. MPU_RGDAACn field descriptions (continued)

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 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 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 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 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

Table 135. MPU_RGDAACn field descriptions (continued)

Field	Description
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 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 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.

18.6 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 bus cycles.

18.6.1 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 140*, the access evaluation macro inputs the system bus address phase signals 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).

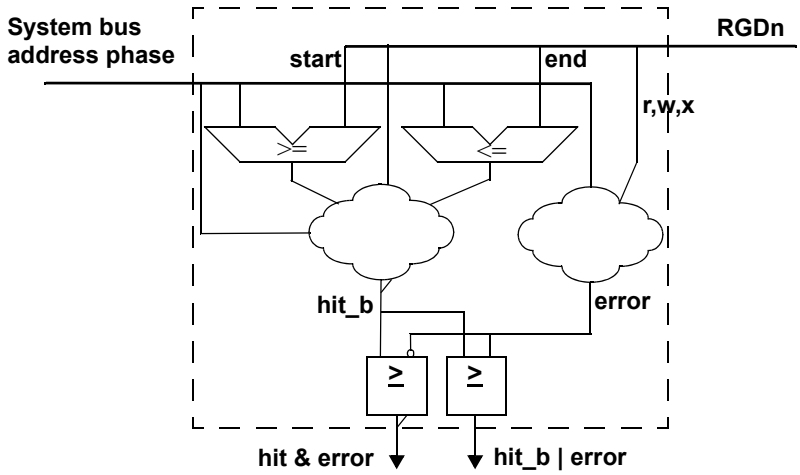


Figure 140. MPU access evaluation macro

Figure 140 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 evaluate the region hit determination, the MPU uses two magnitude comparators in conjunction with the contents of a region descriptor: the current access must be included between the region's “start” and “end” addresses and simultaneously the region's valid bit must be active.

Recall there are no hardware checks to verify that region's “end” address is greater then region's “start” address, and it is software’s responsibility to properly load appropriate values into these fields of the region descriptor.

In addition to this, the optional process identifier is examined against the region descriptor’s PID and PIDMASK fields. In order to generate the pid_hit indication: the current PID with its PIDMASK must be equal to the region's PID with its PIDMASK. Also the process identifier enable is take into account in this comparison so that the MPU forces the pid_hit term to be asserted in the case of AHB bus master doesn't provide its process identifier.

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. The protection violation logic then evaluates the access against the effective permissions using the specification shown in *Table 136*.

Table 136. Protection violation definition

Description	Inputs			Output
	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection violation?
inst fetch read	—	—	0	yes, no x permission
inst fetch read	—	—	1	no, access is allowed
data read	0	—	—	yes, no r permission

Table 136. Protection violation definition (continued)

Description	Inputs			Output
	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection violation?
data read	1	—	—	no, access is allowed
data write	—	0	—	yes, no w permission
data write	—	1	—	no, access is allowed

As shown in *Figure 140*, the output of the protection violation logic is the error signal.

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.

18.6.2 Putting it all together and AHB error terminations

For each XBAR 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 18.8 Application information*.

In event of a protection error, the MPU requires two distinct actions:

1. Intercepting the error during the address phase (first cycle out of two) and cancelling the transaction before it is seen by the slave device
2. Performing the required logic functions to force the standard 2-cycle AHB error response to properly terminate the bus transaction and then providing the right values to the crossbar switch to commit the transaction to other portions of the platform.

If, instead, the access is allowed, then the MPU simply passes all “original” signals to the slave device. In this case, from a functionality point of view, the MPU is fully transparent.

18.7 Initialization information

The reset state of MPU_CESR[VLD] disables the entire module. Recall that, 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) is 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.

18.8 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.

When the MPU detects an access error, the current 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 bus cycle typically initiates some type of error response in the originating bus master. For example, the CPU errors will generate a core exception, whereas the DMA errors will generate a MPU (external) interrupt. It is important to highlight that in case of DMA access violations the core will continue to run, but if a core violation occurs the system will stop. 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].

19 System Integration Unit Lite (SIUL)

19.1 Introduction

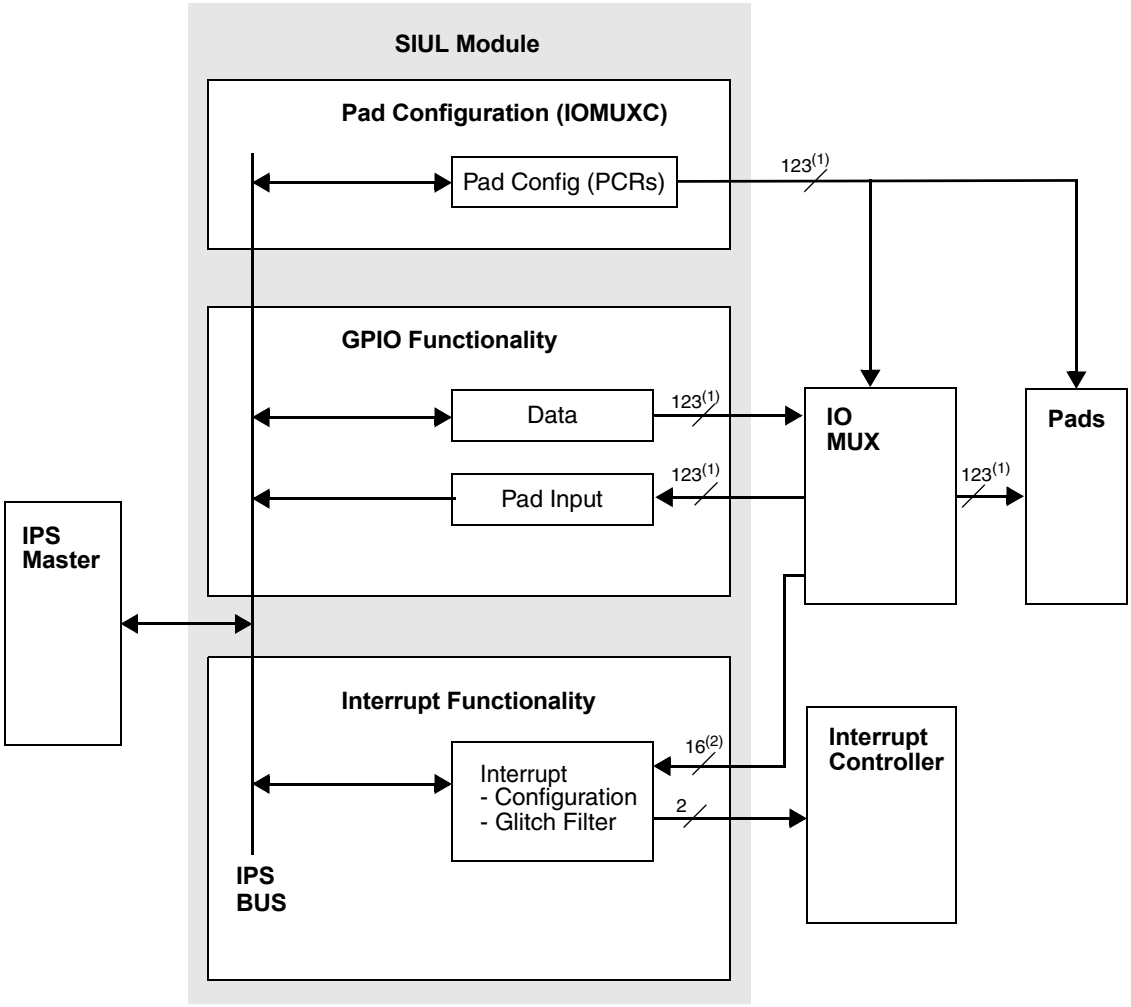
This chapter describes the System Integration Unit Lite (SIUL), which is used for the management of the pads and their configuration. It controls the multiplexing of the alternate functions used on all pads as well as being responsible for the management of the external interrupts to the device.

19.2 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 141* provides a block diagram of the SIUL and its interfaces to other system components.

The module provides the capability to configure, read, and write to the device's general-purpose I/O pads that can be configured as either inputs or outputs.

- When a pad is configured as an input, the state of the pad (logic high or low) is obtained by reading an associated data input register.
- When a pad is configured as an output, the value driven onto the pad is determined by writing to an associated data output register. Enabling the input buffers when a pad is configured as an output allows the actual state of the pad to be read.
- To enable monitoring of an output pad value, the pad can be configured as both output and input so the actual pad value can be read back and compared with the expected value.



Notes:

1. Up to 123 I/O pins in 144-pin and 208-pin packages; up to 79 I/O pins in 100-pin packages
2. Up to 16 I/O pins in 144-pin and 208-pin packages; up to 12 I/O pins in 100-pin packages

Figure 141. System Integration Unit Lite block diagram

19.3 Features

The System Integration Unit Lite supports these distinctive features:

- GPIO
  - GPIO function on up to 123 I/O pins
  - Dedicated input and output registers for most GPIO pins[n]
- External interrupts
  - 2 system interrupt vectors for up to 16 interrupt sources
  - 16 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

19.4 External signal description

Most device pads support multiple device functions. Pad configuration registers are provided to enable selection between GPIO and other signals. These other signals, also referred to as alternate functions, are typically peripheral functions.

GPIO pads are grouped in “ports”, with each port containing up to 16 pads. With appropriate configuration, all pins in a port can be read or written to in parallel with a single R/W access.

Note: In order to use GPIO port functionality, all pads in the port must be configured as GPIO rather than as alternate functions.

Table 137 lists the external pins configurable via the SIUL.

Table 137. SIUL signal properties

GPIO[0:122](1) category	Name	I/O direction	Function
System configuration	GPIO [0:19] [26:47] [60:122]	Input/Output	General-purpose input/output
	GPIO [20:25] [48:59]	Input	Analog precise channels, low power oscillator pins
External interrupt	EIRQ[0:15](2)	Input	Pins with External Interrupt Request functionality. Please see the signal description chapter of this reference manual for details.

1. GPIO[0:122] in 144-pin LQFP and LBGA208; GPIO[0:78] in 100-pin LQFP
2. EIRQ[12:15] available only in 144-pin LQFP

n. Some device pins, e.g., analog pins, do not have both input and output functionality.

19.4.1 Detailed signal descriptions

General-purpose I/O pins (GPIO[0:122])

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.

External interrupt request input pins (EIRQ[0:15])[o]

The EIRQ[0:15] pins are connected to the SIUL inputs. Rising- or falling-edge events are enabled by setting the corresponding bits in the SIUL_IREER or the SIUL_IFEER register.

o. EIRQ[0:15] in 144-pin LQFP and LBGA208 packages; EIRQ[0:11] in the 100-pin LQFP

19.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

19.5.1 SIUL memory map

Table 138 gives an overview of the SIUL registers implemented.

Table 138. SIUL memory map

Base address: 0xC3F9_0000		
Address offset	**Register**	**Location**
0x0000	Reserved	
0x0004	MCU ID Register #1 (MIDR1)	*on page 19-329*
0x0008	MCU ID Register #2 (MIDR2)	*on page 19-330*
0x000C–0x0013	Reserved	
0x0014	Interrupt Status Flag Register (ISR)	*on page 19-331*
0x0018	Interrupt Request Enable Register (IRER)	*on page 19-332*
0x001C–0x0027	Reserved	
0x0028	Interrupt Rising-Edge Event Enable Register (IREER)	*on page 19-332*
0x002C	Interrupt Falling-Edge Event Enable Register (IFEER)	*on page 19-333*
0x0030	Interrupt Filter Enable Register (IFER)	*on page 19-334*
0x0034–0x003F	Reserved	
0x0040–0x0134	Pad Configuration Registers (PCR0–PCR122)(1)	*on page 19-335*
0x0136–0x04FF	Reserved	
0x0500–0x051C	Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI28_31)	*on page 19-337*
0x0520–0x05FF	Reserved	
0x0600–0x0678	GPIO Pad Data Output Registers (GPDO0_3–GPDO120_123)(2),(3)	*on page 19-340*
0x067C–0x07FF	Reserved	
0x0800–0x0878	GPIO Pad Data Input Registers (GPDI0_3–GPDI120_123)(2),(4)	*on page 19-341*
0x087C–0x0BFF	Reserved	
0x0C00–0x0C0C	Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO3)	*on page 19-341*
0x0C10–0x0C3F	Reserved	
0x0C40–0x0C4C	Parallel GPIO Pad Data In Registers (PGPDI0 – PGPDI3)	*on page 19-342*
0x0C50–0x0C7F	Reserved	
0x0C80–0x0C9C	Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO7)	*on page 19-343*
0x0CA0–0x0FFF	Reserved	

Table 138. SIUL memory map (continued)

Base address: 0xC3F9_0000		
Address offset	**Register**	**Location**
0x1000–0x103C	Interrupt Filter Maximum Counter Registers (IFMC0–IFMC15)(5)	*on page 19-344*
0x1040–0x107C	Reserved	
0x1080	Interrupt Filter Clock Prescaler Register (IFCPR)	*on page 19-345*
0x1084–0x3FFF	Reserved	

1. PCR[0:122] is valid in the 144-pin LQFP and the LBGA208 packages, while in the 100-pin LQFP packages is PCR[0:78], so all the remaining registers are reserved.
2. Not all registers are used. The registers, although byte-accessible are allocated on 32-bit boundaries. There are some unused registers at the end of the space. The number of unused registers is further reduced in packages with reduced GPIO pin count.
3. GPDO[0:123] is valid in the 144-pin LQFP and the LBGA208 packages, while in the 100-pin LQFP packages is GPDO[0:76], so all the remaining registers are reserved.
4. GPDI[0:123] is valid in the 144-pin LQFP and the LBGA208 packages, while in the 100-pin LQFP packages is GPDI[0:76], so all the remaining registers are reserved.
5. IFMC[0:15] is valid in the 144-pin LQFP and the LBGA208 packages, while in the 100-pin LQFP packages is IFMC[0:11], so all the remaining registers are reserved.

Note: A transfer error will be issued when trying to access completely reserved register space.

19.5.2 Register protection

Individual registers in System Integration Unit Lite can be protected from accidental writes using the Register Protection module. The following registers can be protected:

- Interrupt Request Enable Register (IRER)
- Interrupt Rising-Edge Event Enable Register (IREER)
- Interrupt Falling-Edge Event Enable Register (IFEER)
- Interrupt Filter Enable Register (IFER),
- Pad Configuration Registers (PCR0–PCR122). Note that only the following registers can be protected:
  - PCR[0:15] (Port A)
  - PCR[16:19] (Port B[0:3])
  - PCR[34:47] (Port C[2:15])
- Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI28_31)
- Interrupt Filter Maximum Counter Registers (IFMC0–IFMC15). Note that only IFMC[0:15] can be protected.
- Interrupt Filter Clock Prescaler Register (IFCPR)

See the “Register Under Protection” appendix for more details.

19.5.3 Register descriptions

MCU ID Register #1 (MIDR1)

This register holds identification information about the device.

Figure 142. MCU ID Register #1 (MIDR1)

Offset: 0x0004 Access: Read

	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	0	0

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

Table 139. MIDR1 field descriptions

Field	Description
PARTNUM[15:0]	MCU Part Number, lower 16 bits Device part number of the MCU. 0101_0110_0000_0001:128 KB 0101_0110_0000_0010: 256 KB 0101_0110_0000_0011: 320/384 KB 0101_0110_0000_0100: 512 KB For the full part number this field needs to be combined with MIDR2[PARTNUM[23:16]].
CSP	Always reads back 0
PKG	Package Settings Can be read by software to determine the package type that is used for the particular device as described below. Any values not explicitly specified are reserved. 0b00001: 64-pin LQFP 0b01001: 100-pin LQFP 0b01101: 144-pin LQFP
MAJOR_MASK	Major Mask Revision Counter starting at 0x0. Incremented each time there is a resynthesis.
MINOR_MASK	Minor Mask Revision Counter starting at 0x0. Incremented each time a mask change is done.

MCU ID Register #2 (MIDR2)

Figure 143. MCU ID Register #2 (MIDR2)

Offset: 0x0008 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SF	FLASH_SIZE_1				FLASH_SIZE_2			0	0	0	0	0	0	0	0
W																
Reset	1	0	1	0	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	PARTNUM[23:16]								0	0	0	EE	0	0	0	0
W																
Reset	0	1	0	0	0	0	1	0/1	0	0	0	1	0(1)	0(1)	0(1)	0

1. Static bit fixed in hardware

Table 140. MIDR2 field descriptions

Field	Description
SF	Manufacturer 0 Reserved 1 ST
FLASH_SIZE_1	Coarse granularity for Flash memory size Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2 0011 128 KB 0100 256 KB 0101 512 KB
FLASH_SIZE_2	Fine granularity for Flash memory size Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2 0000 0 x (FLASH_SIZE_1 / 8) 0010 2 x (FLASH_SIZE_1 / 8) 0100 4 x (FLASH_SIZE_1 / 8)
PARTNUM [23:16]	MCU Part Number, upper 8 bits containing the ASCII character within the MCU part number 0x42h: Character ‘B’ (Body controller) 0x43h: Character ‘C’ (Gateway) For the full part number this field needs to be combined with MIDR1[PARTNUM[15:0]].
EE	Data Flash present 0 No Data Flash is present 1 Data Flash is present

Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

Figure 144. Interrupt Status Flag Register (ISR)

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	EIF[15:0](1)															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. EIF[15:0] in 144-pin LQFP and the LBGA208 packages; EIF[11:0] in 100-pin LQFP package.

Table 141. ISR field descriptions

Field	Description
EIF[x]	External Interrupt Status Flag x This flag can be cleared only by writing a ‘1’. Writing a ‘0’ has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request. 0 No interrupt event has occurred on the pad 1 An interrupt event as defined by IREER[x] and IFEER[x] has occurred

Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging to the interrupt controller.

Figure 145. Interrupt Request Enable Register (IRER)

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 W	IRE[15:0](1)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. IRE[15:0] in 144-pin LQFP and the LBGA208 packages; IRE[11:0] in 100-pin LQFP package.

Table 142. IRER field descriptions

Field	Description
IRE[x]	External Interrupt Request Enable x 0 Interrupt requests from the corresponding ISR[EIF[x]] bit are disabled. 1 Interrupt requests from the corresponding ISR[EIF[x]] bit are enabled.

Interrupt Rising-Edge Event Enable Register (IREER)

This register is used to enable rising-edge triggered events on the corresponding interrupt pads.

Figure 146. Interrupt Rising-Edge Event Enable Register (IREER)

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 W	IREE[15:0](1)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. IREE[15:0] in 144-pin LQFP and LBGA208 packages; IREE[11:0] in 100-pin LQFP package.

Table 143. IREER field descriptions

Field	Description
IREE[x]	Enable rising-edge events to cause the ISR[EIF[x]] bit to be set. 0 Rising-edge event is disabled 1 Rising-edge event is enabled

Interrupt Falling-Edge Event Enable Register (IFEER)

This register is used to enable falling-edge triggered events on the corresponding interrupt pads.

Figure 147. Interrupt Falling-Edge Event Enable Register (IFEER)

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 W	IFEE[15:0](1)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. IFEE[15:0] in 144-pin LQFP and LBGA208 packages; IFEE[11:0] in 100-pin LQFP package.

Table 144. IFEER field descriptions

Field	Description
IFEE[x]	Enable falling-edge events to cause the ISR[EIF[x]] bit to be set. 0 Falling-edge event is disabled 1 Falling-edge event is 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. If IREER[IREE] and IFEER[IFEE] bits are set for the same source the interrupts are triggered by both rising edge events and falling edge events.

Interrupt Filter Enable Register (IFER)

This register is used to enable a digital filter counter on the corresponding interrupt pads to filter out glitches on the inputs.

Figure 148. Interrupt Filter Enable Register (IFER)

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 W	IFE[15:0](1)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. IFE[15:0] in 144-pin LQFP and LBGA208 packages; IFE[11:0] in 100-pin LQFP package.

Table 145. IFER field descriptions

Field	Description
IFE[x]	Enable digital glitch filter on the interrupt pad input 0 Filter is disabled 1 Filter is enabled See the IFMC field descriptions in *Table 156* for details on how the filter works.

Pad Configuration Registers (PCR0–PCR122)

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.

Please note that input and output peripheral muxing are separate.

- For output pads:
  - Select the appropriate alternate function in Pad Config Register (PCR)
  - OBE is not required for functions other than GPIO
- For INPUT pads:
  - Select the feature location from PSMI register
  - Set the IBE bit in the appropriate PCR
- For normal GPIO (not alternate function):
  - Configure PCR
  - Read from GPDI or write to GPDO

Figure 149. Pad Configuration Registers (PCRx)

Offsets: Base + 0x0040 (PCR0)(123 registers)
Base + 0x0042 (PCR1)
...
Base + 0x0130 (PCR122)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC		PA[1:0]		OBE	IBE	0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset	0	0(1)	0	0	0(1)	0	0(2)	0(3)	0	0	0	0	0	0	0(3)	1(4)

1. SMC and PA[1] are ‘1’ for JTAG pads
2. OBE is ‘1’ for TDO
3. IBE and WPE are ‘1’ for TCK, TMS, TDI, FAB and ABS
4. WPS is ‘0’ for input only pad with analog feature and FAB

Note: 16/32-bit access is supported.

In addition to the bit map above, the following *Table 147* describes the PCR depending on the pad type (pad types are defined in the “Pad types” section of this reference manual). 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 146. 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 function)		SMC	APC		PA[1:0]		OBE	IBE			ODE			SRC	WPE	WPS
J (Pad with GPIO and analog functionality)		SMC	APC		PA[1:0]		OBE	IBE			ODE			SRC	WPE	WPS
I (Pad dedicated to ADC)		SMC	APC		PA[1:0]		OBE	IBE			ODE			SRC	WPE	WPS

Table 147. PCRx 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, the output buffer of the pad is disabled. 1 In SAFE mode, the output buffer remains functional.

Table 147. PCRx field descriptions (continued)

Field	Description
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 is used to select the function that is allowed to drive the output of a multiplexed pad. 00 Alternative Mode 0 — GPIO 01 Alternative Mode 1 — See the signal description chapter 10 Alternative Mode 2 — See the signal description chapter 11 Alternative Mode 3 — See the signal description chapter *Note: Number of bits depends on the actual number of actual alternate functions. Please see datasheet.*
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 is disabled when PA[1:0] = 00 1 Output buffer of the pad is enabled when PA[1:0] = 00
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 0 Input buffer of the pad is disabled 1 Input buffer of the pad is 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 Pad configured for push/pull output 1 Pad configured for open drain
SRC	Slew Rate Control This field controls the slew rate of the associated pad when it is slew rate selectable. Its usage is the following: 0 Pad configured as slow (default) 1 Pad is configured as medium or fast (depending on the pad) *Note: PC[1] (TDO pad) is medium only. By default SRC = 0, and writing '1' has no effect.*
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 disabled for the pad 1 Weak pull device enabled 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 Weak pull-down selected 1 Weak pull-up selected

Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI28_31)

In some cases, a peripheral input signal can be selected from more than one pin. For example, the CAN1_RXD signal can be selected on three different pins: PC[3], PC[11] and PF[15]. Only one can be active at a time. To select the pad to be used as input to the peripheral:

- Select the signal via the pad's PCR register using the PA field.
- Specify the pad to be used via the appropriate PSMI field.

Figure 150. Pad Selection for Multiplexed Inputs Register (PSMI0_3)

Offsets:0x0500–0x051C (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	PADSEL0				0	0	0	0	PADSEL1			
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				0	0	0	0	PADSEL3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 148. PSMI0_3 field descriptions

Field	Description
PADSEL0–3, PADSEL4–7, ... PADSEL28–31	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function. See *Table 149*.

In order to multiplex different pads to the same peripheral input, the SIUL provides a register that controls the selection between the different sources.

Table 149. Peripheral input pin selection

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI0_3	PADSEL0	0x500	CAN1RX / FlexCAN_1	00: PCR[35] 01: PCR[43] 10: PCR[95][2]
	PADSEL1	0x501	CAN2RX / FlexCAN_2	00: PCR[73] 01: PCR[89][2]
	PADSEL2	0x502	CAN3RX / FlexCAN_3	00: PCR[36] 01: PCR[73] 10: PCR[89][2]
	PADSEL3[3]	0x503	CAN4RX / FlexCAN_4	00: PCR[35] 01: PCR[43] 10: PCR[95][2]
PSMI4_7	PADSEL4[3]	0x504	CAN5RX / FlexCAN_5	00: PCR[64] 01: PCR[97][2]
	PADSEL5	0x505	SCK_0 / DSPI_0	00: PCR[14] 01: PCR[15]
	PADSEL6	0x506	CS0_0 / DSPI_0	00: PCR[14] 01: PCR[15] 10: PCR[27]
	PADSEL7	0x507	SCK_1 / DSPI_1	00: PCR[34] 01: PCR[68] 10: PCR[114][2]
PSMI8_11	PADSEL8	0x508	SIN_1 / DSPI_1	00: PCR[36] 01: PCR[66] 10: PCR[112][2]
	PADSEL9	0x509	CS0_1 / DSPI_1	00: PCR[435] 01: PCR[61] 10: PCR[69] 11: PCR[115][2]
	PADSEL10	0x50A	SCK_2 / DSPI_2	00: PCR[46] 01: PCR[78][2] 10: PCR[105][2]
	PADSEL11	0x50B	SIN_2 / DSPI_2	00: PCR[44] 01: PCR[76]
PSMI12_15	PADSEL12	0x50C	CS0_2 / DSPI_2	00: PCR[47] 01: PCR[79][2] 10: PCR[82][2] 11: PCR[104][2]
	PADSEL13	0x50D	E1UC[3] / eMIOS_0	00: PCR[3] 01: PCR[27]
	PADSEL14	0x50E	E0UC[4] / eMIOS_0	00: PCR[4] 01: PCR[28]
	PADSEL15	0x50F	E0UC[5] / eMIOS_0	00: PCR[5] 01: PCR[29]

Table 149. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI16_19	PADSEL16	0x510	E0UC[6] / eMIOS_0	00: PCR[6] 01: PCR[30]
	PADSEL17	0x511	E0UC[7] / eMIOS_0	00: PCR[7] 01: PCR[31]
	PADSEL18	0x512	E0UC[10] / eMIOS_0	00: PCR[10] 01: PCR[80][2]
	PADSEL19	0x513	E0UC[11] / eMIOS_0	00: PCR[11] 01: PCR[81][2]
PSMI20_23	PADSEL20	0x514	E0UC[12] / eMIOS_0	00: PCR[44] 01: PCR[82][2]
	PADSEL21	0x515	E0UC[13] / eMIOS_0	00: PCR[45] 01: PCR[83][2]
	PADSEL22	0x516	E0UC[14] / eMIOS_0	00: PCR[46] 01: PCR[84][2]
	PADSEL23	0x517	E0UC[22] / eMIOS_0	00: PCR[70] 01: PCR[72] 10: PCR[85][2]
PSMI24_27	PADSEL24	0x518	E0UC[23] / eMIOS_0	00: PCR[71] 01: PCR[73] 10: PCR[86][2]
	PADSEL25[4]	0x519	E0UC[24] / eMIOS_0	00: PCR[60] 01: PCR[106][2]
	PADSEL26[4]	0x51A	E0UC[25] / eMIOS_0	00: PCR[61] 01: PCR[107][2]
	PADSEL27[4]	0x51B	E0UC[26] / eMIOS_0	00: PCR[62] 01: PCR[108][2]
PSMI28_31	PADSEL28[4]	0x51C	E0UC[27] / eMIOS_0	00: PCR[63] 01: PCR[109][2]
	PADSEL29	0x51D	SCL / f_0	00: PCR[11] 01: PCR[19]
	PADSEL30	0x51E	SDA / I2C__0	00: PCR[10] 01: PCR[18]
	PADSEL31	0x51F	LIN3RX / LINFlex_3	00: PCR[8] 01: PCR[75]

1. See the signal description chapter of this reference manual for correspondence between PCR and pinout
2. Not available in 100-pin LQFP
3. Available only on SPC560B50B2 devices
4. Not available on SPC560B40L3 devices

GPIO Pad Data Output Registers (GPDO0_3–GPDO120_123)

These registers are used to set or clear GPIO pads. Each pad data out bit can be controlled separately with a byte access.

Figure 151. Port GPIO Pad Data Output Register 0–3 (GPDO0_3)

Offsets: 0x0600–0x0678 (31 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	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 150. GPDO0_3 field descriptions

Field	Description
PDO[x]	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

Caution: Toggling several IOs at the same time can significantly increase the current in a pad group. Caution must be taken to avoid exceeding maximum current thresholds. Please see datasheet.

GPIO Pad Data Input Registers (GPDI0_3–GPDI120_123)

These registers are used to read the GPIO pad data with a byte access.

Figure 152. Port GPIO Pad Data Input Register 0–3 (GPDI0_3)

Offsets: 0x0800–0x0878 (31 registers) 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	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 151. GPDI0_3 field descriptions

Field	Description
PDI[x]	Pad Data In This bit stores the value of the external GPIO pad associated with this register. 0 Value of the data in signal for the corresponding GPIO pad is logic low 1 Value of the data in signal for the corresponding GPIO pad is logic high

Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO3)

SPC560Bx and SPC560Cx devices ports are constructed such that they contain 16 GPIO pins, for example PortA[0..15]. Parallel port registers for input (PGPDI) and output (PGPDO) are provided to allow a complete port to be written or read in one operation, dependent on the individual pad configuration.

Writing a parallel PGPDO register directly sets the associated GPDO register bits. There is also a masked parallel port output register allowing the user to determine which pins within a port are written.

While very convenient and fast, this approach does have implications regarding current consumption for the device power segment containing the port GPIO pads. Toggling several GPIO pins simultaneously can significantly increase current consumption.

Caution: Caution must be taken to avoid exceeding maximum current thresholds when toggling multiple GPIO pins simultaneously. Please see datasheet.

Table 152 shows the locations and structure of the PGPDO*x* registers.

Table 152. PGPDO0 – PGPDO3 register map

Offset[1]	Register	Field																														
		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
0x0C00	PGPDO0	Port A																Port B															
0x0C04	PGPDO1	Port C																Port D															

Table 152. PGPDO0 – PGPDO3 register map (continued)

Offset[1]	Register	Field																													
		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
0x0C08	PGPDO2	Port E																Port F														
0x0C0C	PGPDO3	Port G																Port H														

1. SIU base address is 0xC3F9_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in *Table 152*, the PGPDO0 register contains fields for Port A and Port B.

- Bit 0 is mapped to Port A[0], bit 1 is mapped to Port A[1] and so on, through bit 15, which is mapped to Port A[15]
- Bit 16 is mapped to Port B[0], bit 17 is mapped to Port B[1] and so on, through bit 31, which is mapped to Port B[15].

Parallel GPIO Pad Data In Registers (PGPDI0 – PGPDI3)

The SIU_PGPDI registers are similar in operation to the PGPDIO registers, described in the previous section (*Section Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO3)*) but they are used to read port pins simultaneously.

Note: The port pins to be read need to be configured as inputs but even if a single pin within a port has IBE set, then you can still read that pin using the parallel port register. However, this does mean you need to be very careful.

Reads of PGPDI registers are equivalent to reading the corresponding GPDI registers but significantly faster since as many as two ports can be read simultaneously with a single 32-bit read operation.

Table 153 shows the locations and structure of the PGPDI*x* registers. Each 32-bit PGPDI*x* register contains two 16-bit fields, each field containing the values for a separate port.

Table 153. PGPDI0 – PGPDI3 register map

Offset[1]	Register	Field																													
		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
0x0C40	PGPDI0	Port A																Port B														
0x0C44	PGPDI1	Port C																Port D														
0x0C48	PGPDI2	Port E																Port F														
0x0C4C	PGPDI3	Port G																Port H														

1. SIU base address is 0xC3F9_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in *Table 153*, the PGPDI0 register contains fields for Port A and Port B.

- Bit 0 is mapped to Port A[0], bit 1 is mapped to Port A[1] and so on, through bit 15, which is mapped to Port A[15]
- Bit 16 is mapped to Port B[0], bit 17 is mapped to Port B[1] and so on, through bit 31, which is mapped to Port B[15].

Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO7)

The MPGPDO*x* registers are similar in operation to the PGPDO*x* ports described in *Section Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO3)*, but with two significant differences:

- The MPGPDO*x* registers support *masked* port-wide changes to the data out on the pads of the respective port. Masking effectively allows selective bitwise writes to the full 16-bit port.
- Each 32-bit MPGPDO*x* register is associated to only one port.

Note: The MPGPDOx registers may only be accessed with 32-bit writes. 8-bit or 16-bit writes will not modify any bits in the register and will cause a transfer error response by the module. Read accesses return '0'.

Table 154 shows the locations and structure of the MPGPDO*x* registers. Each 32-bit MPGPDO*x* register contains two 16-bit fields ($MASK_x$ and $MPPDO_x$). The MASK field is a bitwise mask for its associated port. The MPPDO0 field contains the data to be written to the port.

Table 154. MPGPDO0 – MPGPDO7 register map

Offset[1]	Register	Field	
		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
0x0C80	MPGPDO0	MASK0 (Port A)	MPPDO0 (Port A)
0x0C84	MPGPDO1	MASK1 (Port B)	MPPDO1 (Port B)
0x0C88	MPGPDO2	MASK2 (Port C)	MPPDO2 (Port C)
0x0C8C	MPGPDO3	MASK3 (Port D)	MPPDO3 (Port D)
0x0C90	MPGPDO4	MASK4 (Port E)	MPPDO4 (Port E)
0x0C94	MPGPDO5	MASK5 (Port F)	MPPDO5 (Port F)
0x0C98	MPGPDO6	MASK6 (Port G)	MPPDO6 (Port G)
0x0C9C	MPGPDO7	MASK7 (Port H)	MPPDO7 (Port H)

1. SIU base address is 0xC3F9_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in *Table 154*, the MPGPDO0 register contains field MASK0, which is the bitwise mask for Port A and field MPPDO0, which contains data to be written to Port A.

- MPGPDO0[0] is the mask bit for Port A[0], MPGPDO0[1] is the mask bit for Port A[1] and so on, through MPGPDO0[15], which is the mask bit for Port A[15]
- MPGPDO0[16] is the data bit mapped to Port A[0], MPGPDO0[17] is mapped to Port A[1] and so on, through MPGPDO0[31], which is mapped to Port A[15].

Table 155. MPGPDO0..MPGPDO7 field descriptions

Field	Description
$MASK_x$ [15:0]	Mask Field Each bit corresponds to one data bit in the $MPPDO_x$ register at the same bit location. 0 Associated bit value in the $MPPDO_x$field is ignored 1 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 bitwise GPIO Pad Data Output Registers (GPDO0_3–GPDO120_123). 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]

Caution: Toggling several IOs at the same time can significantly increase the current in a pad group. Caution must be taken to avoid exceeding maximum current thresholds. Please see datasheet.

Interrupt Filter Maximum Counter Registers (IFMC0–IFMC15)

These registers are used to configure the filter counter associated with each digital glitch filter.

Note: For the pad transition to trigger an interrupt it must be steady for at least the filter period.

Figure 153. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC15)

Offset: 0x1000–0x103C) (16 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	0	0	0	0	0	0	0	0	0	MAXCNTx			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 156. IFMC field descriptions

Field	Description
MAXCNTx	Maximum Interrupt Filter Counter setting Filter Period = T(CK)*MAXCNTx + n*T(CK) Where (n can be –1 to 3) MAXCNTx can be 0 to 15 T(CK): Prescaled Filter Clock Period, which is FIRC clock prescaled to IFCP value T(FIRC): Basic Filter Clock Period: 62.5 ns (f_{FIRC} = 16 MHz)

Interrupt Filter Clock Prescaler Register (IFCPR)

This register is used to configure a clock prescaler which is used to select the clock for all digital filter counters in the SIUL.

Figure 154. Interrupt Filter Clock Prescaler Register (IFCPR)

Offsets: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			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 157. IFCPR field descriptions

Field	Description
IFCP	Interrupt Filter Clock Prescaler setting Prescaled Filter Clock Period = T(FIRC) x (IFCP + 1) T(FIRC) is the fast internal RC oscillator period. IFCP can be 0 to 15.

19.6 Functional description

19.6.1 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 pad configuration registers (PCR*n*, see *Section Pad Configuration Registers (PCR0–PCR122)*)

allow software control of the static electrical characteristics of external pins with a single write. These are used to 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

19.6.2 General purpose input and output pads (GPIO)

The SIUL manages up to 123 GPIO pads organized as ports that can be accessed for data reads and writes as 32, 16 or 8-bit[p].

Note: Ports are organized as groups of 16 GPIO pads, with the exception of Port J, which has 5. A 32-bit R/W operation accesses two ports simultaneously. A 16-bit operation accesses a full port and an 8-bit access either the upper or lower byte of a port.

As shown in *Figure 155*, all port accesses are identical with each read or write being performed only at a different location to access a different port width.

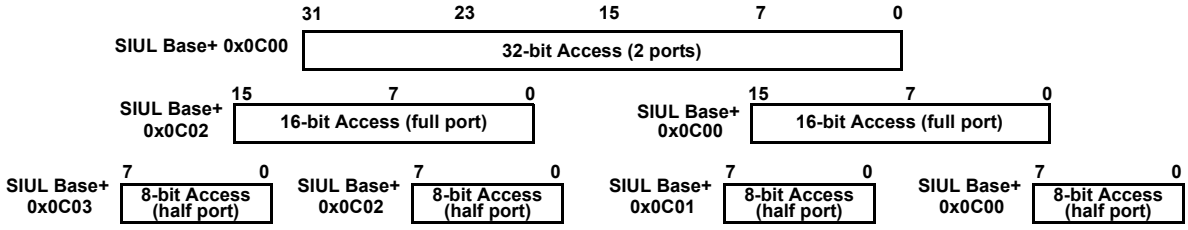


Figure 155. Data Port example arrangement showing configuration for different port width accesses

The SIUL has separate data input (GPDI*n_n*, see *Section GPIO Pad Data Input Registers (GPDI0_3–GPDI120_123)*) and data output (GPDO*n_n*, see *Section GPIO Pad Data Output Registers (GPDO0_3–GPDO120_123)*) 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 simply confirming the value that was written to the data register by accessing the data input registers.

Data output registers allow an output pad to be driven high or low (with the option of push-pull or open drain drive). Input registers are read-only and reflect the respective pad value.

When the pad is configured to use one of its alternate functions, the data input value reflects 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.

p. There are exceptions. Some pads, e.g., precision analog pads, are input only.

The allocation of what input function is connected to the pin is defined by the PSMI registers (PCR*n*, see *Section Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI28_31)*).

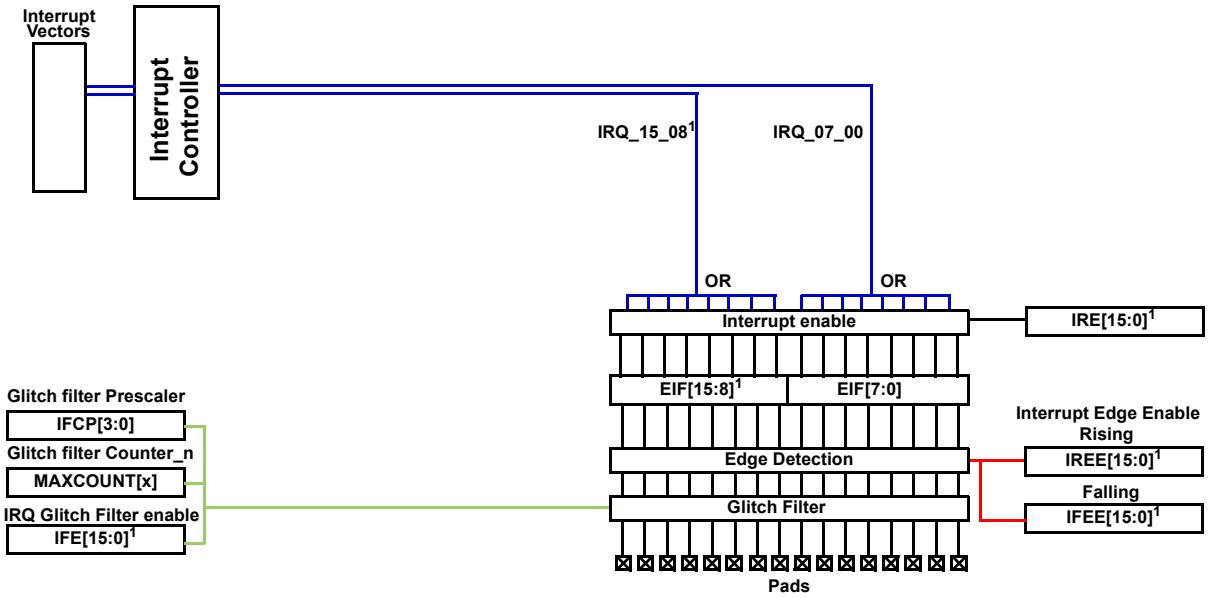
19.6.3 External interrupts

The SIUL supports 16 external interrupts, EIRQ0–EIRQ15. In the signal description chapter of this reference manual, mapping is shown for external interrupts to pads.

The SIUL supports twointerrupt 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.

See *Figure 156* for an overview of the external interrupt implementation.

Figure 156. External interrupt pad diagram



1. This value is valid in the 144-pin LQFP and the 208-pin packages, while there are 12 interrupts in the 100-pin LQFP packages

Each interrupt can be enabled or disabled independently. This can be performed using the 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 which is held in the Interrupt Status Flag Register (ISR). The bits in the ISR[EIF] field are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.

19.7 Pin muxing

For pin muxing, please see the signal description chapter of this reference manual.

20 Inter-Integrated Circuit Bus Controller Module (I^2C)

20.1 Introduction

20.1.1 Overview

The Inter-Integrated Circuit (I^2C™ or IIC) bus is a two wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the number of external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communications over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate up to 100 kbps in Standard Mode and 400 Kbps in Fast Mode. The device is capable of operating at higher baud rates, up to a maximum of module clock/20 with reduced bus loading. Actual baud rate can be less than the programmed baud rate and is dependent on the SCL rise time. SCL rise time is dependent on the external pullup resistor value and bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

20.1.2 Features

The I^2C module has the following key features:

- Compatible with I^2C Bus standard
- Multi-master operation
- Software programmable for one of 256 different serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection

Features currently not supported:

- No support for general call address
- Not compliant to ten-bit addressing

20.1.3 Block diagram

The block diagram of the I^2C module is shown in *Figure 157*.

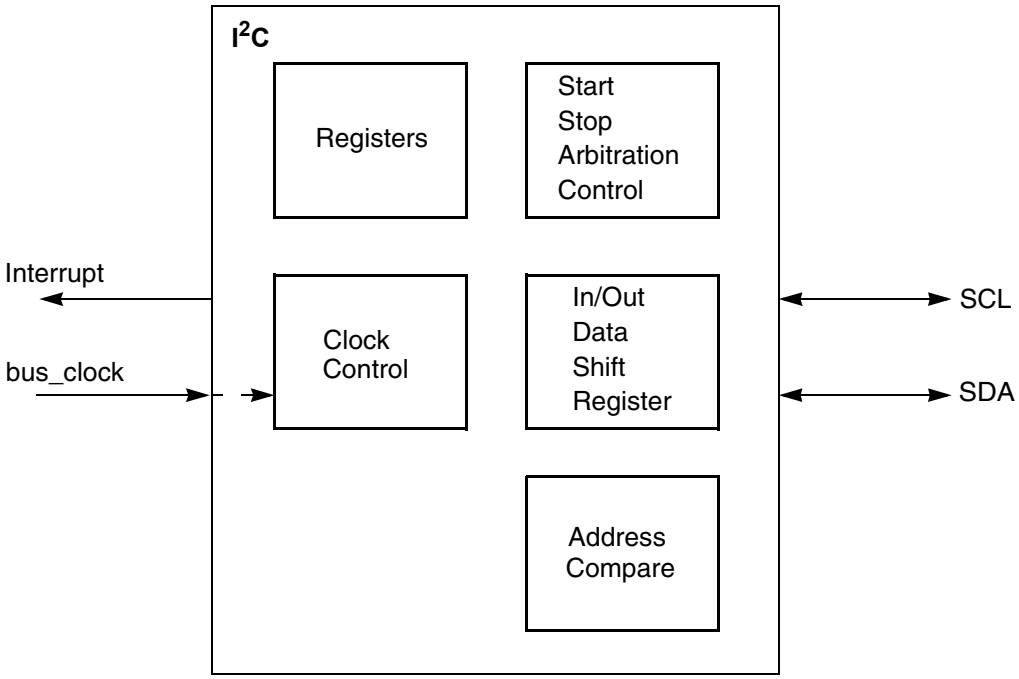


Figure 157. I2C block diagram

20.2 External signal description

The Inter-Integrated Circuit (I2C) module has two external pins, SCL and SDA.

20.2.1 SCL

This is the bidirectional Serial Clock Line (SCL) of the module, compatible with the I2C-Bus specification.

20.2.2 SDA

This is the bidirectional Serial Data line (SDA) of the module, compatible with the I2C-Bus specification.

20.3 Memory map and register description

20.3.1 Module memory map

The memory map for the I2C module is given below in *Table 158*. The total address for each register is the sum of the base address for the I2C module and the address offset for each register.

Table 158. I2C memory map

Base address: 0xFFE3_0000		
Address offset	**Register**	**Location**
0x0	I^2C Bus Address Register (IBAD)	*on page 20-351*
0x1	I^2C Bus Frequency Divider Register (IBFD)	*on page 20-352*
0x2	I^2C Bus Control Register (IBCR)	*on page 20-358*
0x3	I^2C Bus Status Register (IBSR)	*on page 20-359*
0x4	I^2C Bus Data I/O Register (IBDR)	*on page 20-360*
0x5	I^2C Bus Interrupt Config Register (IBIC)	*on page 20-361*

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, the IBDF register for the frequency divider is accessible by a 16-bit read/write to address Base + 0x000, but performing a 16-bit access to Base + 0x001 is illegal.

20.3.2 I^2C Bus Address Register (IBAD)

This register contains the address the I^2C bus will respond to when addressed as a slave; note that it is not the address sent on the bus during the address transfer.

Figure 158. I^2C Bus Address Register (IBAD)

Offset 0x0 Access: Read/write any time

	7	6	5	4	3	2	1	0
R	ADR							0
W								
Reset	0	0	0	0	0	0	0	0

Table 159. IBAD field descriptions

Field	Description
ADR	Slave Address. Specific slave address to be used by the I^2C Bus module. *Note: The default mode of I^2C Bus is slave mode for an address match on the bus.*

20.3.3 I^2C Bus Frequency Divider Register (IBFD)

Figure 159. I^2C Bus Frequency Divider Register (IBFD)

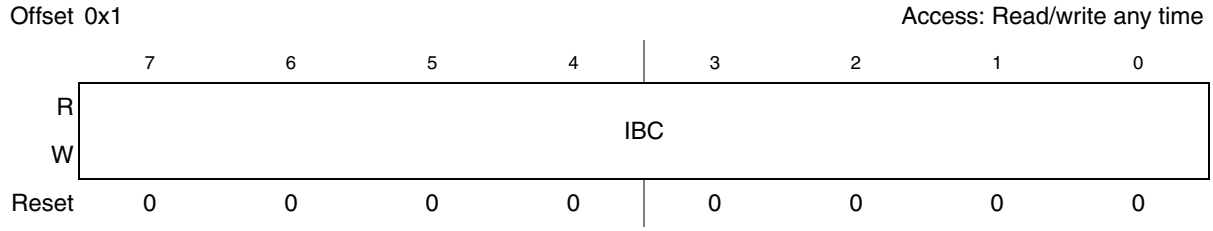


Table 160. IBFD field descriptions

Field	Description
IBC	I-Bus Clock Rate. This field is used to prescale the clock for bit rate selection. The bit clock generator is implemented as a prescale divider. The IBC bits are decoded to give the Tap and Prescale values as follows: 7–6 select the prescaled shift register (see *Table 161*) 5–3 select the prescaler divider (see *Table 162*) 2–0 select the shift register tap point (see *Table 163*)

Table 161. I-Bus multiplier factor

IBC7–6	MUL
00	01
01	02
10	04
11	RESERVED

Table 162. I-Bus prescaler divider values

IBC5–3	scl2start (clocks)	scl2stop (clocks)	scl2tap (clocks)	tap2tap (clocks)
000	2	7	4	1
001	2	7	4	2
010	2	9	6	4
011	6	9	6	8
100	14	17	14	16
101	30	33	30	32
110	62	65	62	64
111	126	129	126	128

Table 163. I-Bus tap and prescale values

IBC2-0	SCL Tap (clocks)	SDA Tap (clocks)
000	5	1
001	6	1
010	7	2
011	8	2
100	9	3
101	10	3
110	12	4
111	15	4

The number of clocks from the falling edge of SCL to the first tap (Tap[1]) is defined by the values shown in the scl2tap column of *Table 162*. All subsequent tap points are separated by 2^{IBC5-3} as shown in the tap2tap column in *Table 162*. The SCL Tap is used to generate the SCL period and the SDA Tap is used to determine the delay from the falling edge of SCL to the change of state of SDA i.e. the SDA hold time.

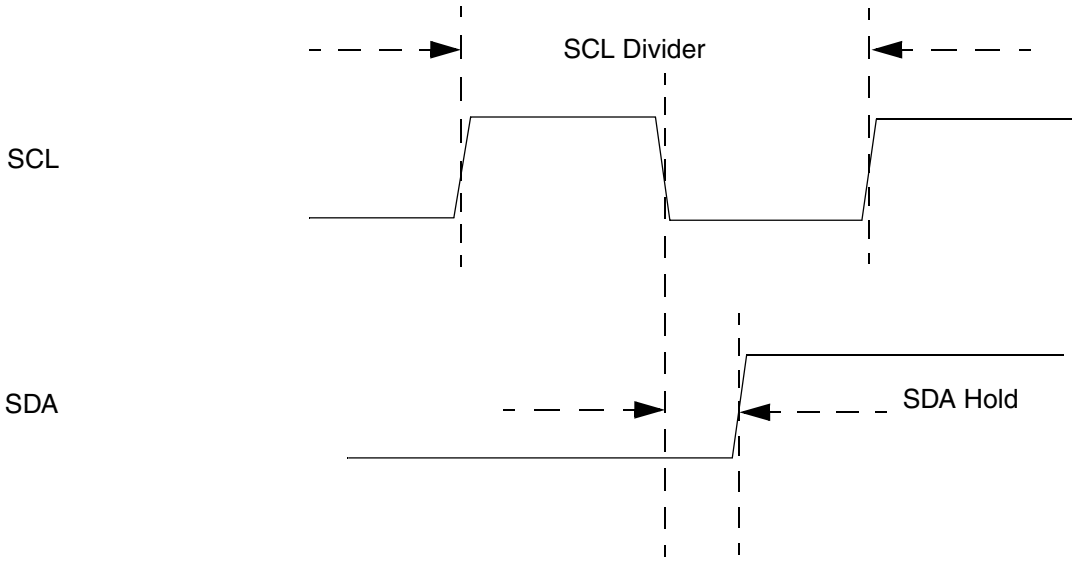


Figure 160. SDA hold time

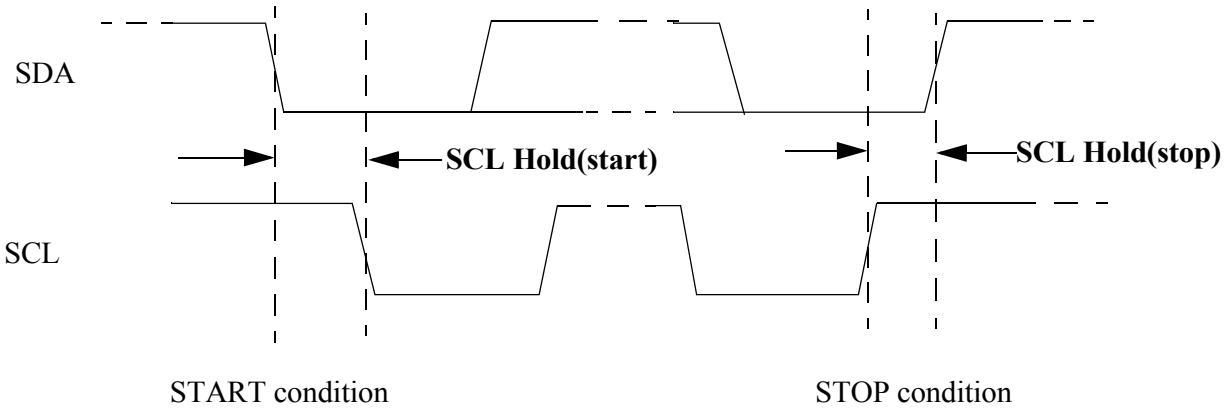


Figure 161. SCL divider and SDA hold

The equation used to generate the divider values from the IBFD bits is:

Equation 3 SCL Divider = MUL x {2 x (scl2tap + [(SCL_Tap -1) x tap2tap] + 2)}

The SDA hold delay is equal to the CPU clock period multiplied by the SDA Hold value shown in *Table 164*. The equation used to generate the SDA Hold value from the IBFD bits is:

Equation 4 SDA Hold = MUL x {scl2tap + [(SDA_Tap - 1) x tap2tap] + 3}

The equation for SCL Hold values to generate the start and stop conditions from the IBFD bits is:

Equation 5 SCL Hold(start) = MUL x [scl2start + (SCL_Tap - 1) x tap2tap]

Equation 6 SCL Hold(stop) = MUL x [scl2stop + (SCL_Tap - 1) x tap2tap]

Table 164. I^2C divider and hold values

	IBC7–0 (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start)	SCL hold (stop)
MUL = 1	00	20	7	6	11
	01	22	7	7	12
	02	24	8	8	13
	03	26	8	9	14
	04	28	9	10	15
	05	30	9	11	16
	06	34	10	13	18
	07	40	10	16	21
	08	28	7	10	15
	09	32	7	12	17
	0A	36	9	14	19
	0B	40	9	16	21
	0C	44	11	18	23
	0D	48	11	20	25
	0E	56	13	24	29
	0F	68	13	30	35
	10	48	9	18	25
	11	56	9	22	29
	12	64	13	26	33
	13	72	13	30	37
	14	80	17	34	41
	15	88	17	38	45
	16	104	21	46	53
	17	128	21	58	65
	18	80	9	38	41
	19	96	9	46	49
	1A	112	17	54	57
	1B	128	17	62	65
	1C	144	25	70	73
	1D	160	25	78	81
	1E	192	33	94	97
	1F	240	33	118	121
	20	160	17	78	81
	21	192	17	94	97
	22	224	33	110	113
	23	256	33	126	129
	24	288	49	142	145
	25	320	49	158	161
	26	384	65	190	193
	27	480	65	238	241
	28	320	33	158	161
	29	384	33	190	193
	2A	448	65	222	225
	2B	512	65	254	257
	2C	576	97	286	289
	2D	640	97	318	321
	2E	768	129	382	385
	2F	960	129	478	481
	30	640	65	318	321
	31	768	65	382	385
	32	896	129	446	449
	33	1024	129	510	513
	34	1152	193	574	577
	35	1280	193	638	641
	36	1536	257	766	769
	37	1920	257	958	961
	38	1280	129	638	641
	39	1536	129	766	769
	3A	1792	257	894	897
	3B	2048	257	1022	1025
	3C	2304	385	1150	1153
	3D	2560	385	1278	1281
	3E	3072	513	1534	1537
	3F	3840	513	1918	1921

Table 164. I^2C divider and hold values

	IBC7–0 (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start)	SCL hold (stop)
MUL = 2	40	40	14	12	22
	41	44	14	14	24
	42	48	16	16	26
	43	52	16	18	28
	44	56	18	20	30
	45	60	18	22	32
	46	68	20	26	36
	47	80	20	32	42
	48	56	14	20	30
	49	64	14	24	34
	4A	72	18	28	38
	4B	80	18	32	42
	4C	88	22	36	46
	4D	96	22	40	50
	4E	112	26	48	58
	4F	136	26	60	70
	50	96	18	36	50
	51	112	18	44	58
	52	128	26	52	66
	53	144	26	60	74
	54	160	34	68	82
	55	176	34	76	90
	56	208	42	92	106
	57	256	42	116	130
	58	160	18	76	82
	59	192	18	92	98
	5A	224	34	108	114
	5B	256	34	124	130
	5C	288	50	140	146
	5D	320	50	156	162
	5E	384	66	188	194
	5F	480	66	236	242
	60	320	28	156	162
	61	384	28	188	194
	62	448	32	220	226
	63	512	32	252	258
	64	576	36	284	290
	65	640	36	316	322
	66	768	40	380	386
	67	960	40	476	482
	68	640	28	316	322
	69	768	28	380	386
	6A	896	36	444	450
	6B	1024	36	508	514
	6C	1152	44	572	578
	6D	1280	44	636	642
	6E	1536	52	764	770
	6F	1920	52	956	962
	70	1280	36	636	642
	71	1536	36	764	770
	72	1792	52	892	898
	73	2048	52	1020	1026
	74	2304	68	1148	1154
	75	2560	68	1276	1282
	76	3072	84	1532	1538
	77	3840	84	1916	1922
	78	2560	36	1276	1282
	79	3072	36	1532	1538
	7A	3584	68	1788	1794
	7B	4096	68	2044	2050
	7C	4608	100	2300	2306
	7D	5120	100	2556	2562
	7E	6144	132	3068	3074
	7F	7680	132	3836	3842

Table 164. I^2C divider and hold values

	IBC7–0 (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start)	SCL hold (stop)
MUL = 4	80	80	28	24	44
	81	88	28	28	48
	82	96	32	32	52
	83	104	32	36	56
	84	112	36	40	60
	85	120	36	44	64
	86	136	40	52	72
	87	160	40	64	84
	88	112	28	40	60
	89	128	28	48	68
	8A	144	36	56	76
	8B	160	36	64	84
	8C	176	44	72	92
	8D	192	44	80	100
	8E	224	52	96	116
	8F	272	52	120	140
	90	192	36	72	100
	91	224	36	88	116
	92	256	52	104	132
	93	288	52	120	148
	94	320	68	136	164
	95	352	68	152	180
	96	416	84	184	212
	97	512	84	232	260
	98	320	36	152	164
	99	384	36	184	196
	9A	448	68	216	228
	9B	512	68	248	260
	9C	576	100	280	292
	9D	640	100	312	324
	9E	768	132	376	388
	9F	960	132	472	484
	A0	640	68	312	324
	A1	768	68	376	388
	A2	896	132	440	452
	A3	1024	132	504	516
	A4	1152	196	568	580
	A5	1280	196	632	644
	A6	1536	260	760	772
	A7	1920	260	952	964
	A8	1280	132	632	644
	A9	1536	132	760	772
	AA	1792	260	888	900
	AB	2048	260	1016	1028
	AC	2304	388	1144	1156
	AD	2560	388	1272	1284
	AE	3072	516	1528	1540
	AF	3840	516	1912	1924
	30	2560	260	1272	1284
	B1	3072	260	1528	1540
	B2	3584	516	1784	1796
	B3	4096	516	2040	2052
	B4	4608	772	2296	2308
	B5	5120	772	2552	2564
	B6	6144	1028	3064	3076
	B7	7680	1028	3832	3844
	B8	5120	516	2552	2564
	B9	6144	516	3064	3076
	BA	7168	1028	3576	3588
	BB	8192	1028	4088	4100
	BC	9216	1540	4600	4612
	BD	10240	1540	5112	5124
	BE	12288	2052	6136	6148
	BF	15360	2052	7672	7684

20.3.4 I^2C Bus Control Register (IBCR)

Figure 162. I^2C Bus Control Register (IBCR)

Offset 0x2 Access: Read/write any time

	7	6	5	4	3	2	1	0
R	MDIS	IBIE	MSSL	TXRX	NOACK	0	DMAEN	0
W						RSTA		
Reset	1	0	0	0	0	0	0	0

Table 165. IBCR field descriptions

Field	Description
MDIS	Module disable. This bit controls the software reset of the entire I^2C Bus module. 1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can still be accessed. Status register bits (IBSR) are not valid when module is disabled. 0 The I^2C Bus module is enabled. This bit must be cleared before any other IBCR bits have any effect *Note: If the I^2C Bus module is enabled in the middle of a byte transfer, the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating whenever a subsequent start condition is detected. Master mode will not be aware that the bus is busy, hence if a start cycle is initiated then the current bus cycle may become corrupt. This would ultimately result in either the current bus master or the I^2C Bus module losing arbitration, after which, bus operation would return to normal.*
IBIE	I-Bus Interrupt Enable. 1 Interrupts from the I^2C Bus module are enabled. An I^2C Bus interrupt occurs provided the IBIF bit in the status register is also set. 0 Interrupts from the I^2C Bus module are disabled. Note that this does not clear any currently pending interrupt condition
MSSL	Master/Slave mode select. Upon reset, this bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated only if the IBIF flag is set. MSSL is cleared without generating a STOP signal when the master loses arbitration. 1 Master Mode 0 Slave Mode
TXRX	Transmit/Receive mode select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high. 1 Transmit 0 Receive
NOACK	Data Acknowledge disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I^2C module will always acknowledge address matches, provided it is enabled, regardless of the value of NOACK. Note that values written to this bit are only used when the I^2C Bus is a receiver, not a transmitter. 1 No acknowledge signal response is sent (i.e., acknowledge bit = 1) 0 An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte of data

Table 165. IBCR field descriptions (continued)

Field	Description
RSTA	Repeat Start. Writing a 1 to this bit will generate a repeated START condition on the bus, provided it is the current bus master. This bit will always be read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, will result in loss of arbitration. 1 Generate repeat start cycle 0 No effect
DMAEN	DMA Enable. When this bit is set, the DMA TX and RX lines will be asserted when the I^2C module requires data to be read or written to the data register. No Transfer Done interrupts will be generated when this bit is set, however an interrupt will be generated if the loss of arbitration or addressed as slave conditions occur. The DMA mode is only valid when the I^2C module is configured as a Master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA Application Information section for more details. 1 Enable the DMA TX/RX request signals 0 Disable the DMA TX/RX request signals

20.3.5 I^2C Bus Status Register (IBSR)

Figure 163. I^2C Bus Status Register (IBSR)

Offset 0x3 Access: Read-write

	7	6	5	4	3	2	1	0
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W				w1c			w1c	
Reset	1	0	0	0	0	0	0	0

Table 166. IBSR Field Descriptions

Field	Description
TCF	Transfer complete. While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. Note that this bit is only valid during or immediately following a transfer to the I^2C module or from the I^2C module. 1 Transfer complete 0 Transfer in progress
IAAS	Addressed as a slave. When its own specific address (I-Bus Address Register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU needs to check the SRW bit and set its Tx/$\overline{Rx}$ mode accordingly. Writing to the I-Bus Control Register clears this bit. 1 Addressed as a slave 0 Not addressed
IBB	Bus busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state. 1 Bus is busy 0 Bus is Idle

Table 166. IBSR Field Descriptions (continued)

Field	Description
IBAL	Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: – SDA is sampled low when the master drives a high during an address or data transmit cycle. – SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle. – A start cycle is attempted when the bus is busy. – A repeated start cycle is requested in slave mode. – A stop condition is detected when the master did not request it.
SRW	Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is only valid when the I-Bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master. 1 Slave transmit, master reading from slave 0 Slave receive, master writing to slave
IBIF	I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: – Arbitration lost (IBAL bit set) – Byte transfer complete (TCF bit set - Check w/ design if this is the case (only TCF)) – Addressed as slave (IAAS bit set) – NoAck from Slave (MS & Tx bits set) – I2C Bus going idle (IBB high-low transition and enabled by BIIE) A processor interrupt request will be caused if the IBIE bit is set.
RXAK	Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. This bit is valid only after transfer is complete. 1 No acknowledge received 0 Acknowledge received

20.3.6 I2C Bus Data I/O Register (IBDR)

Figure 164. I2C Bus Data I/O Register (IBDR)

Offset 0x4 Access: Read/write any time

	7	6	5	4	3	2	1	0
R W	DATA							
Reset	0	0	0	0	0	0	0	0

In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. Note that the IBCR[TXRX] field must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I2C is

configured for master transmit but a master receive is desired, then reading the IBDR will not initiate the receive.

Reading the IBDR will return the last byte received while the I^2C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I^2C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master transmit mode, the first byte of data written to IBDR following assertion of MS/$\overline{SL}$ is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/$\overline{W}$ bit (in position D0).

20.3.7 I^2C Bus Interrupt Config Register (IBIC)

Figure 165. I^2C Bus Interrupt Config Register (IBIC)

Offset 0x5 Access: Read/write any time

	7	6	5	4	3	2	1	0
R	BIIE(1)	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

1. This bit cannot be set in reset state, when I2C is in slave mode. It can be set to 1 only when I2C is in Master mode. This information is missing from the spec.

Table 167. IBIC field descriptions

Field	Description
BIIE	Bus Idle Interrupt Enable bit. This config bit can be used to enable the generation of an interrupt once the I^2C bus becomes idle. Once this bit is set, an IBB high-low transition will set the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I^2C bus. 1 Bus Idle Interrupts enabled 0 Bus Idle Interrupts disabled

20.4 Functional description

20.4.1 I-Bus protocol

The I^2C Bus system uses a Serial Data line (SDA) and a Serial Clock Line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. A logical AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer and STOP signal. They are described briefly in the following sections and illustrated in *Figure 166*.

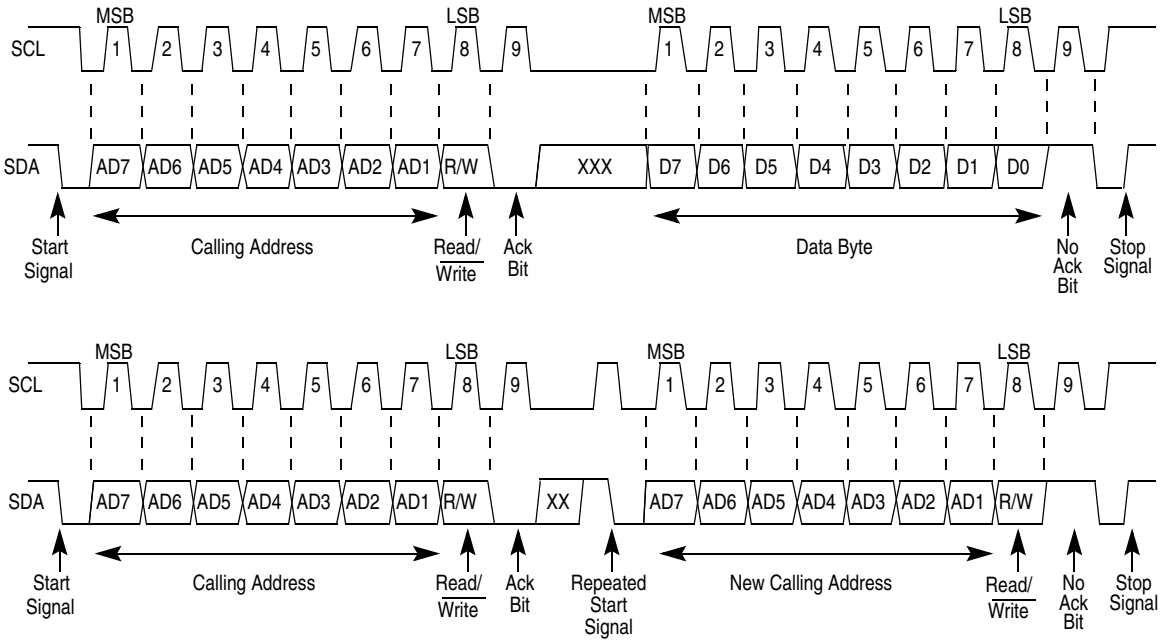


Figure 166. I2C bus transmission signals

START signal

When the bus is free, i.e. no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in *Figure 166*, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

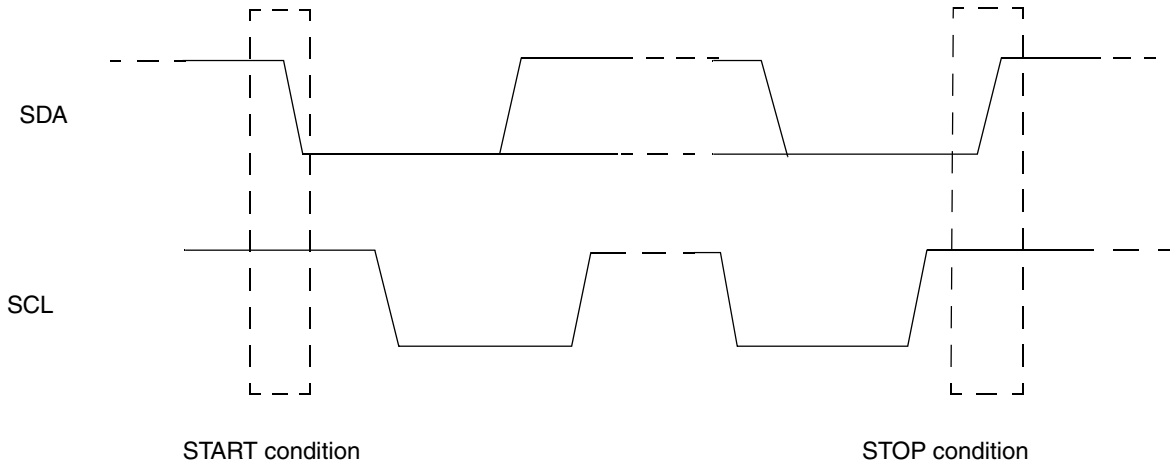


Figure 167. Start and stop conditions

Slave address transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer - the slave transmits data to the master

0 = Write transfer - the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see *Figure 166*).

No two slaves in the system may have the same address. If the I^2C Bus is master, it must not transmit an address that is equal to its own slave address. The I^2C Bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle the I^2C Bus will revert to slave mode and operate correctly, even if it is being addressed by another master.

Data transfer

Once successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in *Figure 166*. There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means 'end of data' to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

STOP signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical “1” (see *Figure 166*).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

Repeated START signal

As shown in *Figure 166*, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

Arbitration procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure. A bus master loses arbitration if it transmits logic “1” while another master transmits logic “0”. The losing masters immediately switch over to slave receive mode and stop driving the SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

Clock synchronization

Since wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see *Figure 168*). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

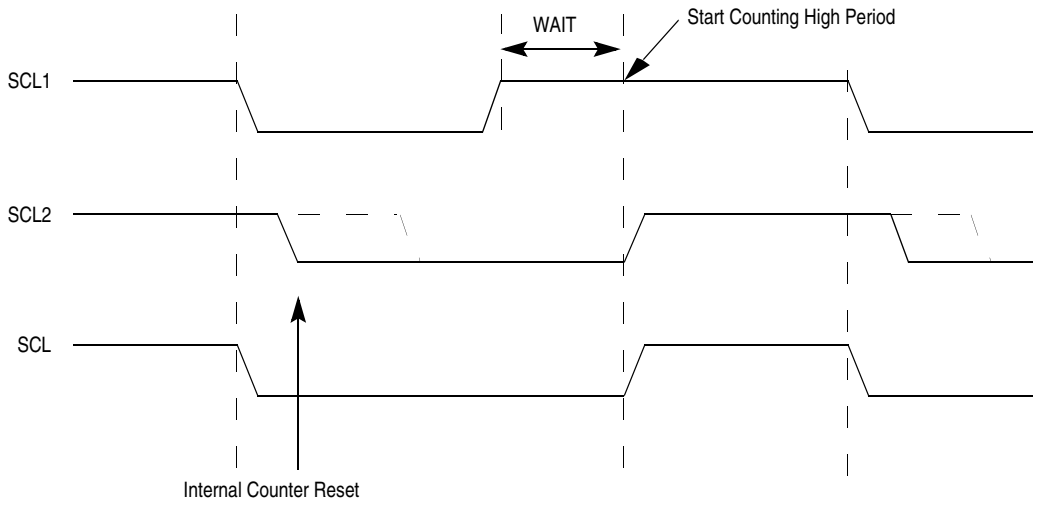


Figure 168. I2C bus clock synchronization

Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

Clock stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

20.4.2 Interrupts

General

The I^2C uses only one interrupt vector.

Table 168. Interrupt summary

Interrupt	Offset	Vector	Priority	Source	Description
I^2C Interrupt	—	—	—	IBAL, TCF, IAAS, IBB bits in IBSR register	When any of IBAL, TCF or IAAS bits is set an interrupt may be caused based on Arbitration lost, Transfer Complete or Address Detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle.

Interrupt description

There are five types of internal interrupts in the I^2C. The interrupt service routine can determine the interrupt type by reading the Status Register.

I^2C Interrupt can be generated on

- Arbitration Lost condition (IBAL bit set)
- Byte Transfer condition (TCF bit set and DMAEN bit not set)
- Address Detect condition (IAAS bit set)
- No Acknowledge from slave received when expected
- Bus Going Idle (IBB bit not set)

The I^2C interrupt is enabled by the IBIE bit in the I^2C Control Register. It must be cleared by writing '1' to the IBIF bit in the interrupt service routine. The Bus Going Idle interrupt needs to be additionally enabled by the BIIE bit in the IBIC register.

20.5 Initialization/application information

20.5.1 I^2C programming examples

Initialization sequence

Reset will put the I^2C Bus Control Register to its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the Frequency Divider Register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I2C Bus Address Register (IBAD) to define its slave address.
3. Clear the IBCR[MDIS] field to enable the I2C interface system.
4. Modify the bits of the I2C Bus Control Register (IBCR) to select Master/Slave mode, Transmit/Receive mode and interrupt enable or not. Optionally also modify the bits of the I2C Bus Interrupt Config Register (IBIC) to further refine the interrupt behavior.

Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' mode. If the device is connected to a multi-master bus system, the state of the I2C Bus Busy bit (IBB) must be tested to check whether the serial bus is free.

If the bus is free (IBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I2C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 5, IBSR ==1)// wait in loop for IBB flag to clear
bit4 and bit 5, IBCR = 1// set transmit and master mode, i.e. generate
start condition
IBDR = calling_address// send the calling address to the data register
while (bit 5, IBSR ==0)// wait in loop for IBB flag to be set
```

Post-transfer software response

Transmission or reception of a byte will set the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I2C Bus interrupt bit (IBIF) is set also; an interrupt will be generated if the interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit will be cleared to indicate data transfer in progress whenever data register is written to in transmit mode, or during reading out from data register in receive mode. The TCF bit should not be used as a data transfer complete flag as the flag timing is dependent on a number of factors including the I2C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. It is recommended that transfer complete situations are detected using the IBIF flag

Software may service the I2C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Note that polling should monitor the IBIF bit rather than the TCF bit since their operation is different when arbitration is lost.

Note that when a “Transfer Complete” interrupt occurs at the end of the address cycle, the master will always be in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit sent with slave calling address, then the Tx/Rx bit at Master side should be toggled at this stage. If Master does not receive an ACK from Slave, then transmission must be re-initiated or terminated.

In slave mode, IAAS bit will get set in IBSR if Slave address (IBAD) matches the Master calling address. This is an indication that Master-Slave data communication can now start. During address cycles (IAAS=1), the SRW bit in the status register is read to determine the direction of the subsequent transfer and the Tx/Rx bit is programmed accordingly. For slave mode data cycles (IAAS=0), the SRW bit is not valid. The Tx/Rx bit in the control register should be read to determine the direction of the current transfer.

Transmit/receive sequence

Follow this sequence in case of Master Transmit(Address/Data):

1. Clear IBSR[IBIF].
2. Write data in Data Register (IBDR).
3. IBSR[TCF] bit will get cleared when transfer is in progress.
4. IBSR[TCF] bit will get set when transfer is complete.
5. Wait for IBSR[IBIF] to get set, then read IBSR register to determine its source:
   - TCF = 1 i.e. transfer is complete.
   - No Acknowledge condition (RXAK = 1) is found.
   - IBB = 0 i.e. Bus has transitioned from Busy to Idle state.
   - If IBB = 1, ignore check of Arbitration Loss (IBAL = 1).
   - Ignore Address Detect (IAAS = 1) for Master mode (valid only for Slave mode).
6. Check RXAK in IBSR for an acknowledge from slave.

Follow this sequence in case of Slave Receive(Address/Data):

1. Clear IBSR[IBIF].
2. IBSR[TCF] will get cleared when transfer is in progress for address transfer.
3. IBSR[TCF] will get set when transfer is complete.
4. Wait for IBSR[IBIF] to get set. Then read IBSR register to determine its source:
   - Address Detect has occurred (IAAS = 1) - determination of Slave mode.
5. Clear IBIF.
6. Wait until IBSR[TCF] bit gets cleared (that is, "Transfer under Progress" condition is reached for data transfer).
7. Wait until IBSR[TCF] bit gets cleared(proof that Transfer Completes from "Transfer under Progress" state).
8. Wait until IBSR[IBIF] bit gets set. To find its source, check if:
   - TCF = 1 i.e. reception is complete
   - IBSR[IBB] = 0, that is, bus has transitioned from Busy to Idle state
   - Ignore Arbitration Loss (IBAL = 1) for IBB = 1
   - Ignore No Acknowledge condition (RXAK = 1) for receiver
9. Read the Data Register (IBDR) to determine data received from Master.

Sequence followed in case of Slave Transmit (Steps 1–4 of Slave Receive for Address Detect, followed by 1–6 of Master Transmit for Data Transmit).

Sequence followed in case of Master Receive (Steps 1–6 of Master Transmit for Address dispatch, followed by 5–8 of Slave Receive for Data Receive).

Generation of STOP

A data transfer ends with a STOP signal generated by the 'master' device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following is an example showing how a stop condition is generated by a master transmitter.

```
if (tx_count == 0) or// check to see if all data bytes have been
transmitted
    (bit 0, IBSR == 1) {// or if no ACK generated
         clear bit 5, IBCR// generate stop condition
        }
else {
IBDR = data_to_transmit// write byte of data to DATA register
         tx_count --// decrement counter
     }// return from interrupt
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data which can be done by setting the transmit acknowledge bit (TXAK) before reading the 2nd last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following is an example showing how a STOP signal is generated by a master receiver.

```
rx_count --// decrease the rx counter
if (rx_count ==1)// 2nd last byte to be read ?
   bit 3, IBCR = 1// disable ACK
if (rx_count == 0)// last byte to be read ?
  bit 5, IBCR = 0// generate stop signal
else
data_received = IBDR// read RX data and store
```

Generation of repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
bit 2, IBCR = 1// generate another start ( restart)
IBDR == calling_address// transmit the calling address
```

Slave mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received. If IAAS is set, software should set the transmit/receive mode select bit (Tx/Rx bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. Note that the only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer may now be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave will drive SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an 'end of data' signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so that the master can generate a STOP signal.

Arbitration lost

If several masters try to engage the bus simultaneously, only one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL=1 and MS/SL=0. If one master attempts to start transmission, while the bus is being engaged by another master, the hardware will inhibit the transmission, switch the MS/SL bit from 1 to 0 without generating a STOP condition, generate an interrupt to CPU and set the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

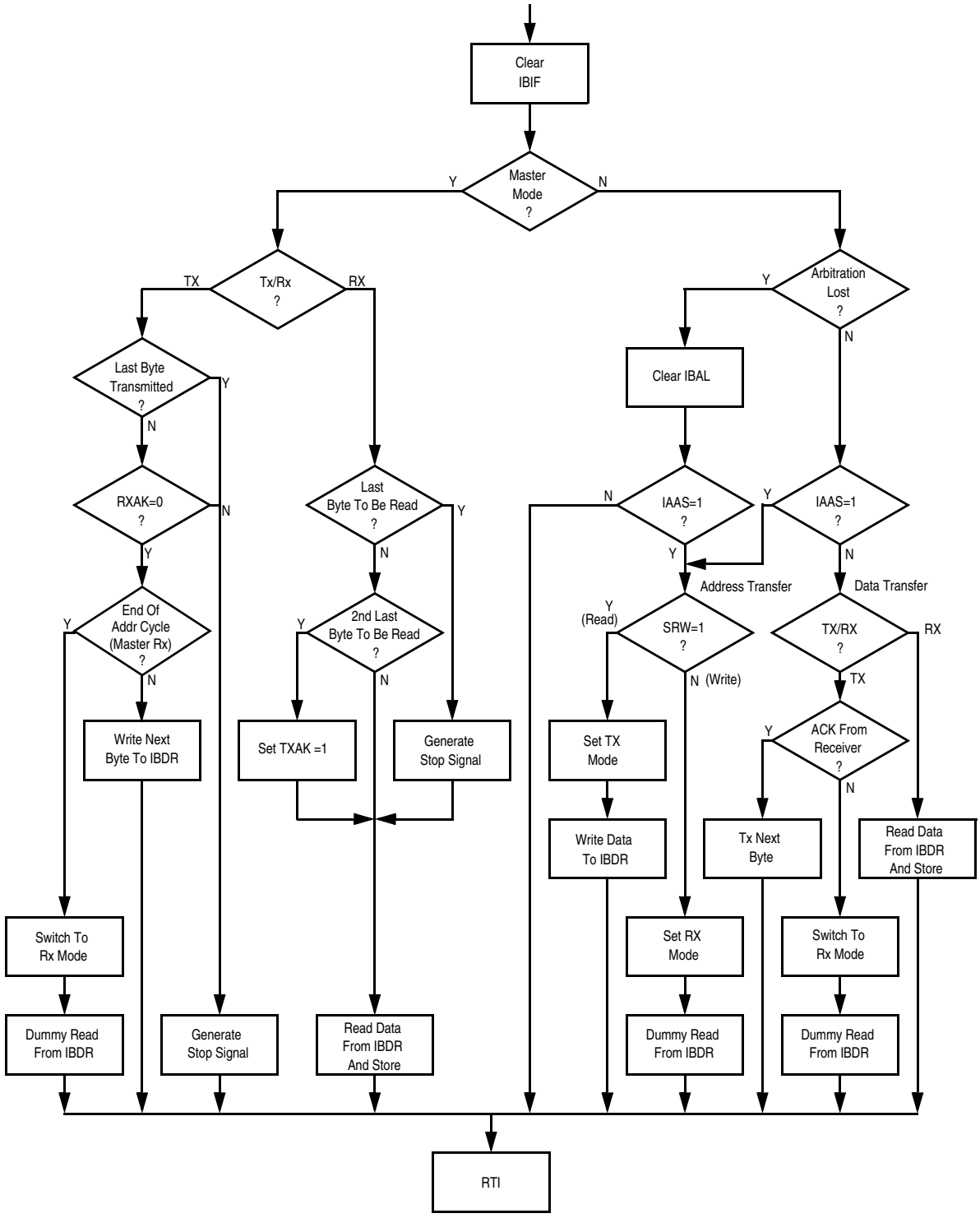


Figure 169. Flow-Chart of Typical I²C Interrupt Routine

21 LIN Controller (LINFlex)

21.1 Introduction

The LINFlex (Local Interconnect Network Flexible) controller interfaces the LIN network and supports the LIN protocol versions 1.3; 2.0 and 2.1; and J2602 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.

21.2 Main features

21.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1 and J2602
- 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[q]
  - 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[q]

21.2.2 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

q. Only LINFlex0 supports slave mode.

21.2.3 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

21.3 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.

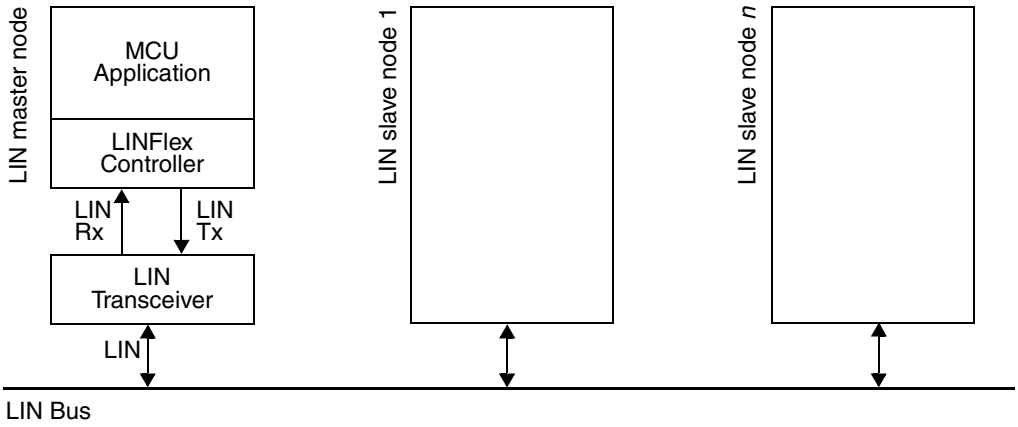
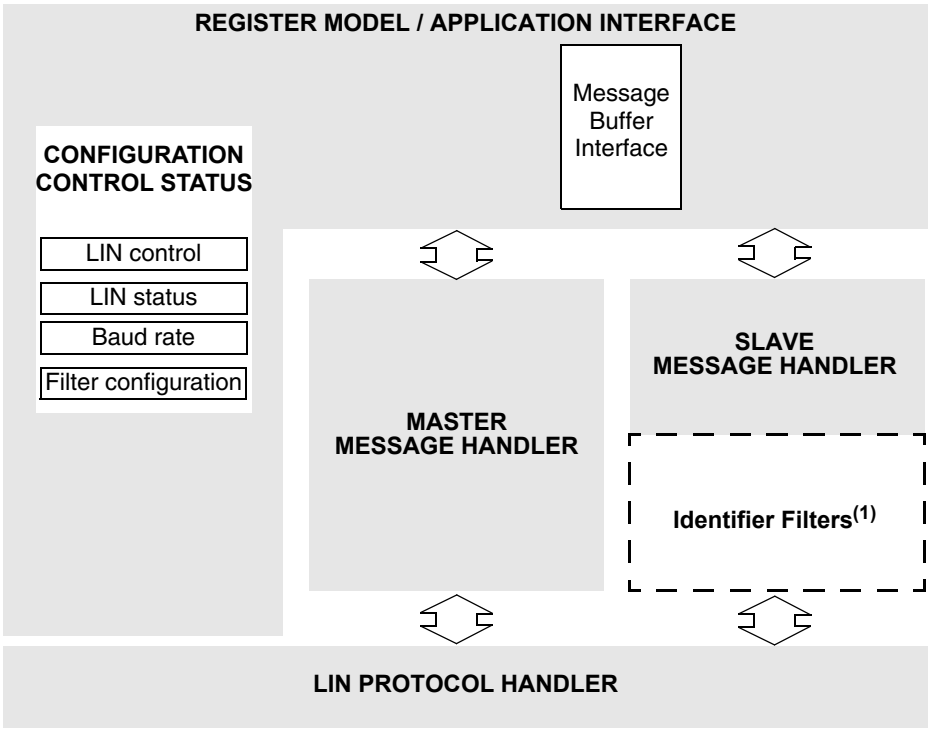


Figure 170. LIN topology network



1. Filter activation optional

Figure 171. LINFlex block diagram

21.4 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 7

$$\text{Tx/ Rx baud} = \frac{f_{periph\_set\_1\_clk}}{(16 \times LFDIV)}$$

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 1 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 2 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 169. 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

21.5 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.

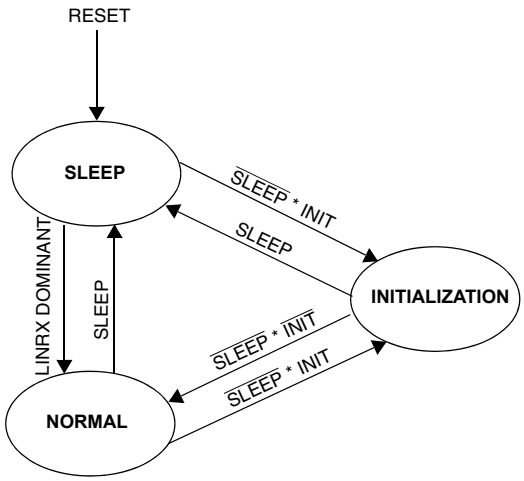


Figure 172. LINFlex operating modes

21.5.1 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.

21.5.2 Normal mode

Once initilization is complete, software clears the INIT bit in the LINCR1 to put the hardware into Normal mode.

21.5.3 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.

21.6 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.

21.6.1 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.

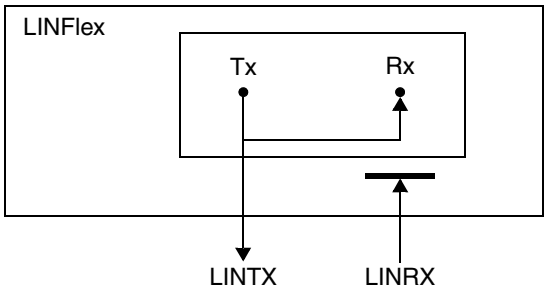


Figure 173. LINFlex in loop back mode

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.

21.6.2 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.

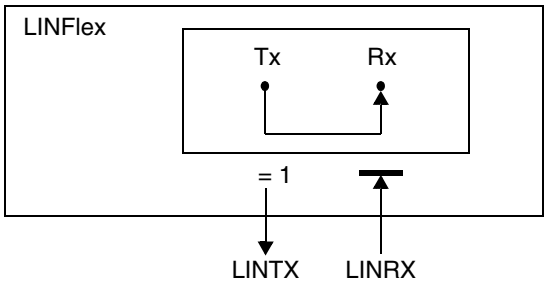


Figure 174. LINFlex in self test mode

21.7 Memory map and registers description

21.7.1 Memory map

See the “Memory map” chapter of this reference manual for the base addresses for the LINFlex modules.

Table 170 shows the LINFlex memory map.

Table 170. LINFlex memory map

Address offset	Register	Location
0x0000	LIN control register 1 (LINCR1)	*on page 21-378*
0x0004	LIN interrupt enable register (LINIER)	*on page 21-381*
0x0008	LIN status register (LINSR)	*on page 21-383*
0x000C	LIN error status register (LINESR)	*on page 21-386*
0x0010	UART mode control register (UARTCR)	*on page 21-387*
0x0014	UART mode status register (UARTSR)	*on page 21-389*
0x0018	LIN timeout control status register (LINTCSR)	*on page 21-391*
0x001C	LIN output compare register (LINOCR)	*on page 21-392*
0x0020	LIN timeout control register (LINTOCR)	*on page 21-392*
0x0024	LIN fractional baud rate register (LINFBRR)	*on page 21-393*
0x0028	LIN integer baud rate register (LINIBRR)	*on page 21-394*
0x002C	LIN checksum field register (LINCFR)	*on page 21-395*
0x0030	LIN control register 2 (LINCR2)	*on page 21-395*
0x0034	Buffer identifier register (BIDR)	*on page 21-397*
0x0038	Buffer data register LSB (BDRL)(1)	*on page 21-398*
0x003C	Buffer data register MSB (BDRM)(2)	*on page 21-398*
0x0040	Identifier filter enable register (IFER)	*on page 21-399*

Table 170. LINFlex memory map (continued)

Address offset	Register	Location
0x0044	Identifier filter match index (IFMI)	on page 21-400
0x0048	Identifier filter mode register (IFMR)	on page 21-401
0x004C	Identifier filter control register 0 (IFCR0)	on page 21-402
0x0050	Identifier filter control register 1 (IFCR1)	on page 21-403
0x0054	Identifier filter control register 2 (IFCR2)	on page 21-403
0x0058	Identifier filter control register 3 (IFCR3)	on page 21-403
0x005C	Identifier filter control register 4 (IFCR4)	on page 21-403
0x0060	Identifier filter control register 5 (IFCR5)	on page 21-403
0x0064	Identifier filter control register 6 (IFCR6)	on page 21-403
0x0068	Identifier filter control register 7 (IFCR7)	on page 21-403
0x006C	Identifier filter control register 8 (IFCR8)	on page 21-403
0x0070	Identifier filter control register 9 (IFCR9)	on page 21-403
0x0074	Identifier filter control register 10 (IFCR10)	on page 21-403
0x0078	Identifier filter control register 11 (IFCR11)	on page 21-403
0x007C	Identifier filter control register 12 (IFCR12)	on page 21-403
0x0080	Identifier filter control register 13 (IFCR13)	on page 21-403
0x0084	Identifier filter control register 14 (IFCR14)	on page 21-403
0x0088	Identifier filter control register 15 (IFCR15)	on page 21-403
0x008C–0x000F	Reserved	

1. LSB: Least significant byte
2. MSB: Most significant byte

LIN control register 1 (LINCR1)

Figure 175. 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 171. LINCR1 field descriptions

Field	Description
CCD	Checksum calculation disable This bit disables the checksum calculation (see *Table 172*). 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 172*). 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. *Note: 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 173*). *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. *Note:* – If no filter is activated, this bit is reserved and always reads 1. – 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 21.6.2, 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.*
LBKM	Loop Back Mode This bit controls the Loop Back mode. For more details see *Section 21.6.1, 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*

Table 171. LINCR1 field descriptions (continued)

Field	Description
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 174*).
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 174*).

Table 172. 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 173. LIN master break length selection

MBL	Length
0000	10-bit
0001	11-bit
0010	12-bit
0011	13-bit
0100	14-bit
0101	15-bit
0110	16-bit
0111	17-bit

Table 173. LIN master break length selection (continued)

MBL	Length
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 174. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

LIN interrupt enable register (LINIER)

Figure 176. 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 175. 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.

Table 175. LINIER field descriptions (continued)

Field	Description
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.
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 177. 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	w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Table 176. 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 176. 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 176. 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 178. 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 177. 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 177. 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 179. 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 178. 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 180. 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 179. 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 179. 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. *Note: 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 181. 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	1	0	0	0	0	0	0	0	0	0

Table 180. 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 182. 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] = 1, this field is read-only.

Table 181. 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 183. 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 182. 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 184. 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 183. 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 185. 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 184. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa This field defines the LINFlex divider (LFDIV) mantissa value (see *Table 185*). This field can be written in Initialization mode only.

Table 185. 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 186. 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 186. 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 172*.

LIN control register 2 (LINCR2)

Figure 187. 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 187. 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 188. 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 188. 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 189. 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 189. 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 190. 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 190. 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 191. 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	0	0	0	0	0	0	0	0	FACT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 191. IFER field descriptions

Field	Description
FACT	Filter activation (see *Table 192*) 0 Filters 2*n* and 2*n* + 1 are deactivated. 1 Filters 2*n* and 2*n* + 1 are activated. This field can be set/cleared in Initialization mode only.

Table 192. IFER[FACT] configuration

Bit	Value	Result
FACT[0]	0	Filters 0 and 1 are deactivated.
	1	Filters 0 and 1 are activated.
FACT[1]	0	Filters 2 and 3 are deactivated.
	1	Filters 2 and 3 are activated.

Table 192. IFER[FACT] configuration (continued)

Bit	Value	Result
FACT[2]	0	Filters 4 and 5 are deactivated.
	1	Filters 4 and 5 are activated.
FACT[3]	0	Filters 6 and 7 are deactivated.
	1	Filters 6 and 7 are activated.
FACT[4]	0	Filters 8 and 9 are deactivated.
	1	Filters 8 and 9 are activated.
FACT[5]	0	Filters 10 and 11 are deactivated.
	1	Filters 10 and 11 are activated.
FACT[6]	0	Filters 12 and 13 are deactivated.
	1	Filters 12 and 13 are activated.
FACT[7]	0	Filters 14 and 15 are deactivated.
	1	Filters 14 and 15 are activated.

Identifier filter match index (IFMI)

Figure 192. 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 193. 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 193. 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 194. IFMR field descriptions

Field	Description
IFM	Filter mode (see *Table 195*). 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 195. 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).

Table 195. IFMR[IFM] configuration (continued)

Bit	Value	Result
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 194. 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											w1c					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: This register can be written in Initialization mode only.

Table 196. 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 195. 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											w1c					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: This register can be written in Initialization mode only.

Table 197. 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

21.8 Functional description

21.8.1 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.

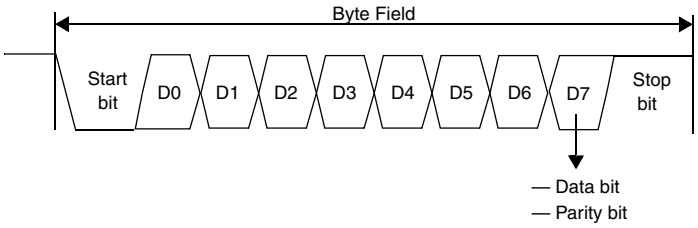


Figure 196. UART mode 8-bit data frame

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.

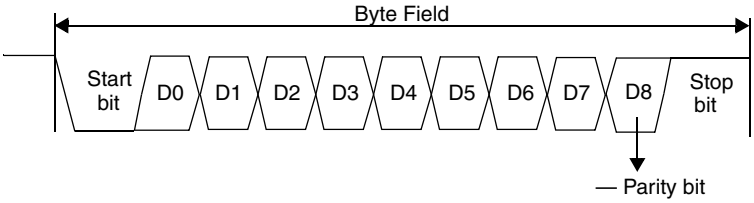


Figure 197. UART mode 9-bit data frame

Buffer in UART mode

The 8-byte buffer is divided into two parts: one for receiver and one for transmitter as shown in *Table 198*.

Table 198. 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 178*).

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.

21.8.2 LIN mode

LIN mode comprises four submodes:

- Master mode
- Slave mode[r]
- Slave mode with identifier filtering[r]
- Slave mode with automatic resynchronization[r]

These submodes are described in the following pages.

r. Only LINFlex0 supports slave mode

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 21.8.3, 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.

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 BDAR. 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 BDAR (see *Figure 199*).

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 BDAR 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 BDAR 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 BDAR to the SRAM (see *Figure 199*).

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 message buffer is full, the next valid message reception leads to an overrun and a message is lost. The hardware sets the BOF bit in the LINSR to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- 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] = 0) the most recent message is discarded and the previous message is available in the buffer.

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 eight 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 198*.

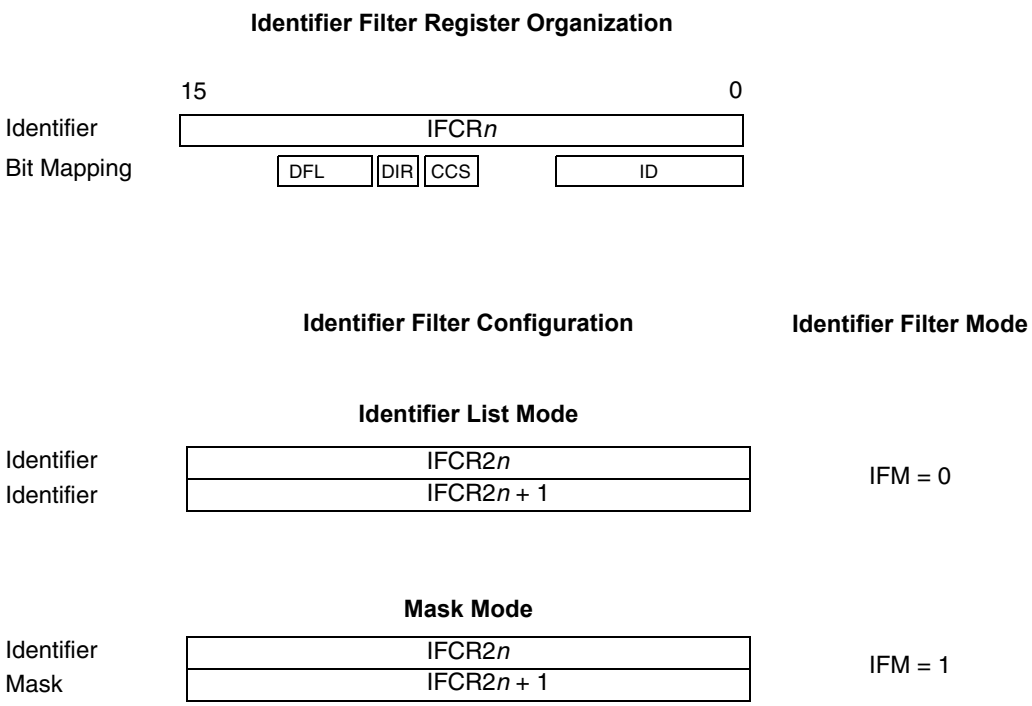


Figure 198. Filter configuration—register organization

Identifier filter mode configuration

The identifier filters are configured in the IFCR*x* registers. To configure an identifier filter the filter must first be deactivated by programming IFER[FACT] = 0.. 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 199. 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
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

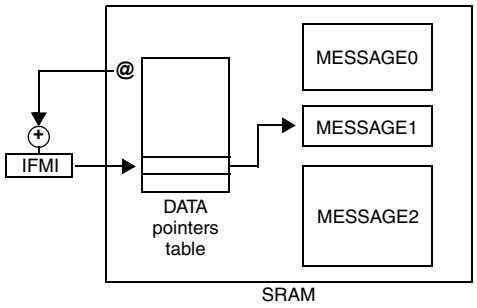


Figure 199. Identifier match index

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 200*). Then the LFDIV value (and its associated registers LINIBRR and LINFBRR) are 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.

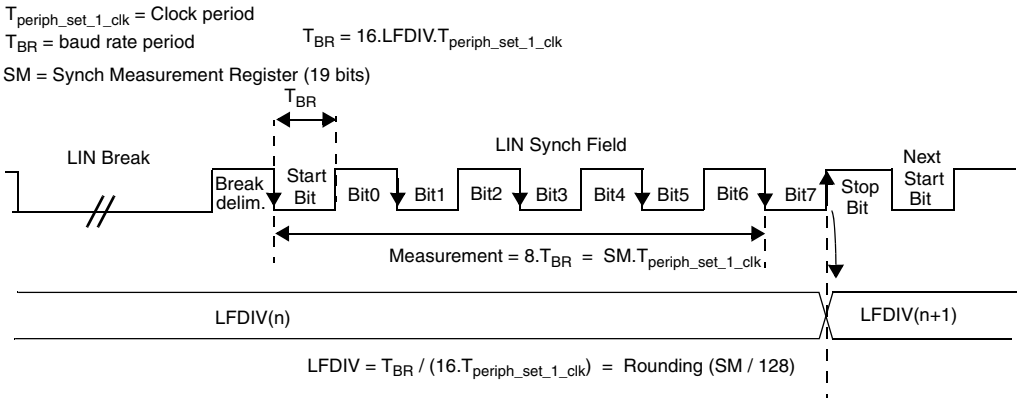


Figure 200. LIN synch field measurement

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.

21.8.3 8-bit timeout counter

LIN timeout mode

Setting 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 201*).

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 201*).

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.

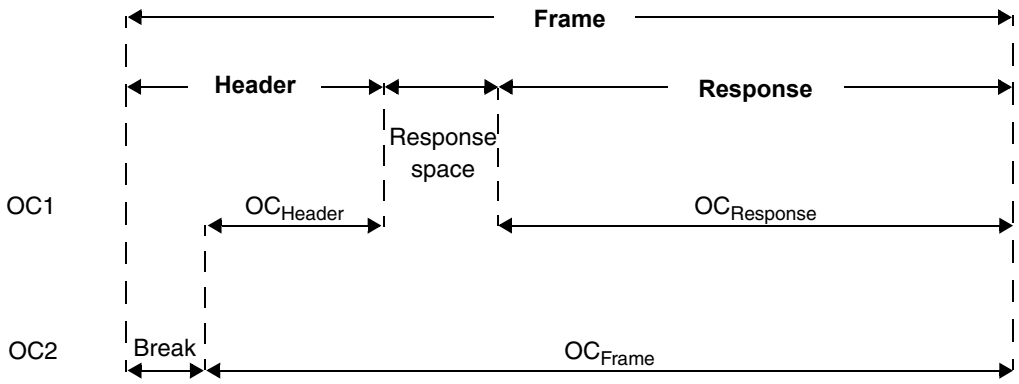


Figure 201. Header and response timeout

Output compare mode

Programming LINTCSR[LTOM] = 0 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.

21.8.4 Interrupts

Table 200. 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
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

22 FlexCAN

22.1 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in *Figure 202*, which describes the main sub-blocks 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 up to 64 Message Buffers is provided. The functions of the submodules are described in subsequent sections.

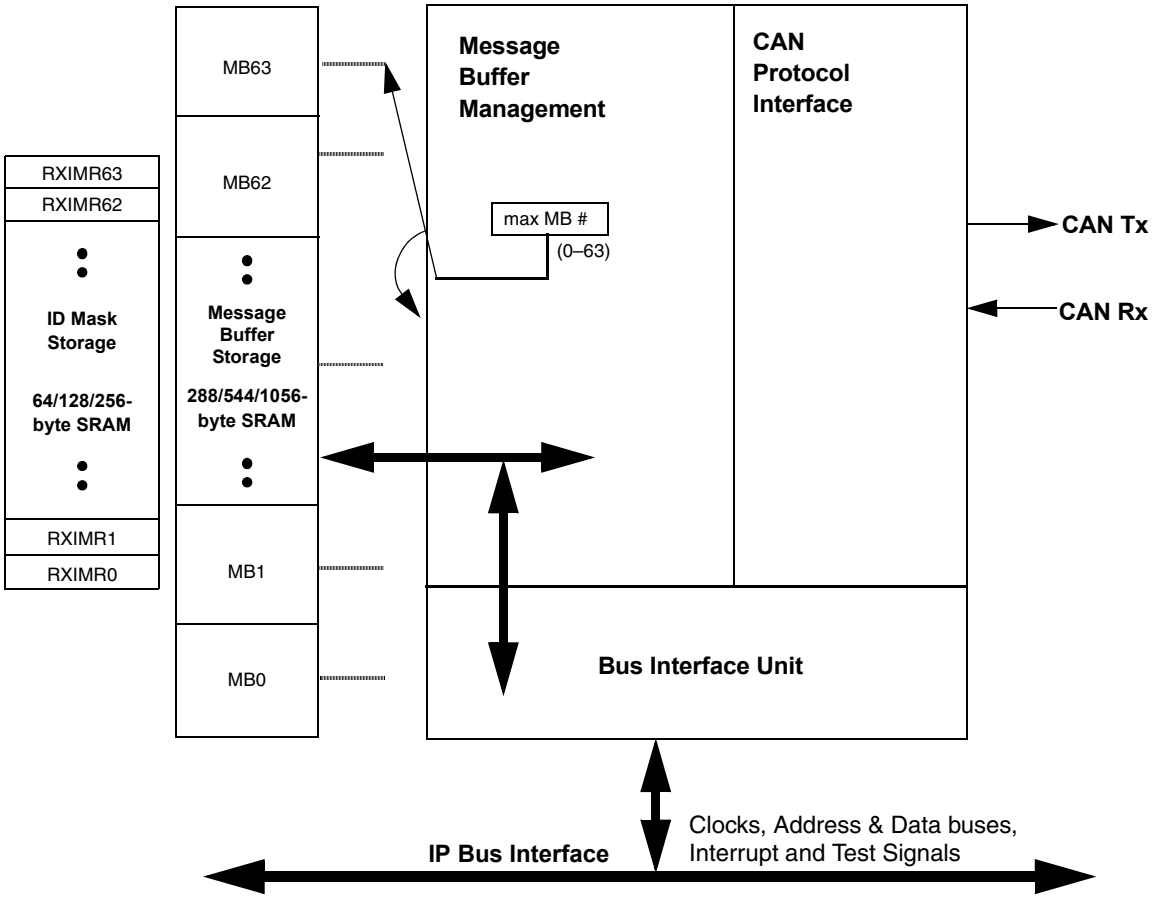


Figure 202. FlexCAN block diagram

22.1.1 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. A flexible number of Message

Buffers (16, 32 or 64) is also supported. The Message Buffers are stored in an embedded SRAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) submodule manages the serial communication on the CAN bus, requesting SRAM 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.

22.1.2 FlexCAN module features

The FlexCAN module includes these distinctive features:

- Full implementation of the CAN protocol specification, version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - 0–8 bytes data length
  - Programmable bit rate up to 1 Mbit/s
  - Content-related addressing
- Flexible Message Buffers (up to 64) of zero to eight bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes either 1056 bytes (64 MBs) of SRAM used for MB storage
- Includes either 256 bytes (64 MBs) of SRAM 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 bits) 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 SRAM 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 mode
- Hardware cancellation on Tx message buffers

22.1.3 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 is also a low power mode: Disable 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 MCR is asserted. If enabled, Freeze Mode is entered when the HALT bit in 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. 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.

22.2 External signal description

22.2.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 201* and described in more detail in the next subsections.

Table 201. FlexCAN signals

Signal name(1)	Direction	Description
CAN Rx	Input	CAN receive pin
CAN Tx	Output	CAN transmit pin

1. The actual MCU pins may have different names.

22.2.2 Signal descriptions

CAN Rx

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’.

CAN Tx

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’.

22.3 Memory map and register 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 MCU. 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 SRAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded SRAM starting at address 0x0880.

22.3.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 64 MBs capability is shown in *Table 202*. Each individual register is identified by its complete name and the corresponding mnemonic. The access type can be Supervisor (S) or Unrestricted (U). Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the SUPV bit in the MCR. These registers are identified as S/U in the Access column of *Table 202*.

The IFLAG2 and IMASK2 registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. 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 MCR is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by SRAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it 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 MCR is

negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Table 202. FlexCAN memory map

Base addresses: 0xFFFC_0000 (FlexCAN_0) 0xFFFC_4000 (FlexCAN_1) 0xFFFC_8000 (FlexCAN_2) 0xFFFC_C000 (FlexCAN_3) 0xFFFD_0000 (FlexCAN_4) 0xFFFD_4000 (FlexCAN_5)		
Address offset	**Register**	**Location**
0x0000	Module Configuration (MCR)	*on page 22-426*
0x0004	Control Register (CTRL)	*on page 22-430*
0x0008	Free Running Timer (TIMER)	*on page 22-433*
0x000C	Reserved	
0x0010	Rx Global Mask (RXGMASK)	*on page 22-434*
0x0014	Rx Buffer 14 Mask (RX14MASK)	*on page 22-435*
0x0018	Rx Buffer 15 Mask (RX15MASK)	*on page 22-435*
0x001C	Error Counter Register (ECR)	*on page 22-436*
0x0020	Error and Status Register (ESR)	*on page 22-437*
0x0024	Interrupt Masks 2 (IMASK2)	*on page 22-440*
0x0028	Interrupt Masks 1 (IMASK1)	*on page 22-441*
0x002C	Interrupt Flags 2 (IFLAG2)	*on page 22-442*
0x0030	Interrupt Flags 1 (IFLAG1)	*on page 22-443*
0x0034–0x005F	Reserved	
0x0060–0x007F	Reserved	
0x0080–0x017F	Message Buffers MB0–MB15	
0x0180–0x027F	Message Buffers MB16–MB31	
0x0280–0x047F	Message Buffers MB32–MB63	
0x0480–0x087F	Reserved	
0x0880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	*on page 22-444*
0x08C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	*on page 22-444*
0x0900–0x097F	Rx Individual Mask Registers RXIMR32–RXIMR63	*on page 22-444*

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 203*. *Table 203* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 203. Message buffer MB0 memory mapping

Address offset	MB field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

22.3.2 Message buffer structure

The Message Buffer structure used by the FlexCAN module is represented in *Table 203*. 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 203. Message Buffer Structure

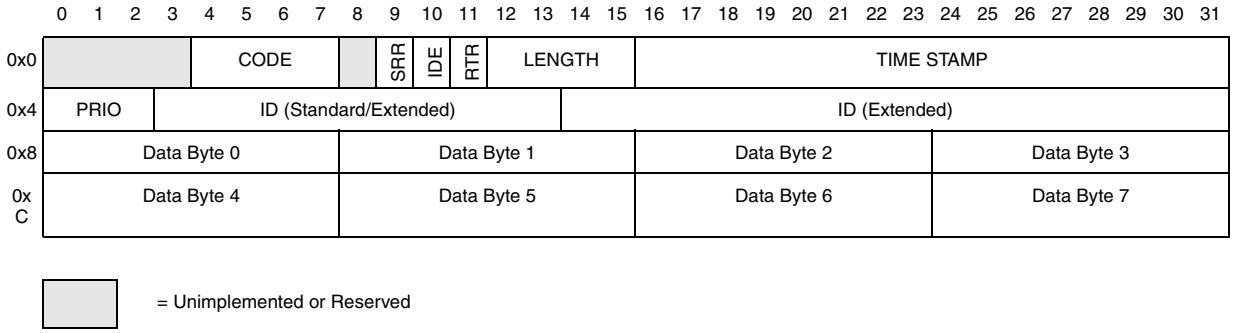


Table 204. Message Buffer Structure field description

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 205* and *Table 206*. See *Section 22.4 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. 1 = Recessive value is compulsory for transmission in Extended Format frames 0 = Dominant is not a valid value for transmission in Extended Format frames
IDE	**ID Extended Bit** This bit identifies whether the frame format is standard or extended. 1 = Frame format is extended 0 = Frame format is standard

Table 204. Message Buffer Structure field description (continued)

Field	Description
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. 1 = Indicates the current MB has a Remote Frame to be transmitted 0 = Indicates the current MB has a Data 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 203*). 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.
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 22.4.4 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** Up to eight 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 205. 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 22.4.6 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 22.4.6 Matching process* for details about overrun behavior.
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 206*), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR.

Table 206. 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.

Table 206. Message buffer code for Tx buffers (continued)

RTR	Initial Tx code	Code after successful transmission	Description
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.

22.3.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *Table 204* shows the Rx FIFO data structure. The region 0x80–0x8C contains an MB structure which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0–0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. *Table 205* 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 22.4.8 Rx FIFO* for more information.

Figure 204. Rx FIFO Structure

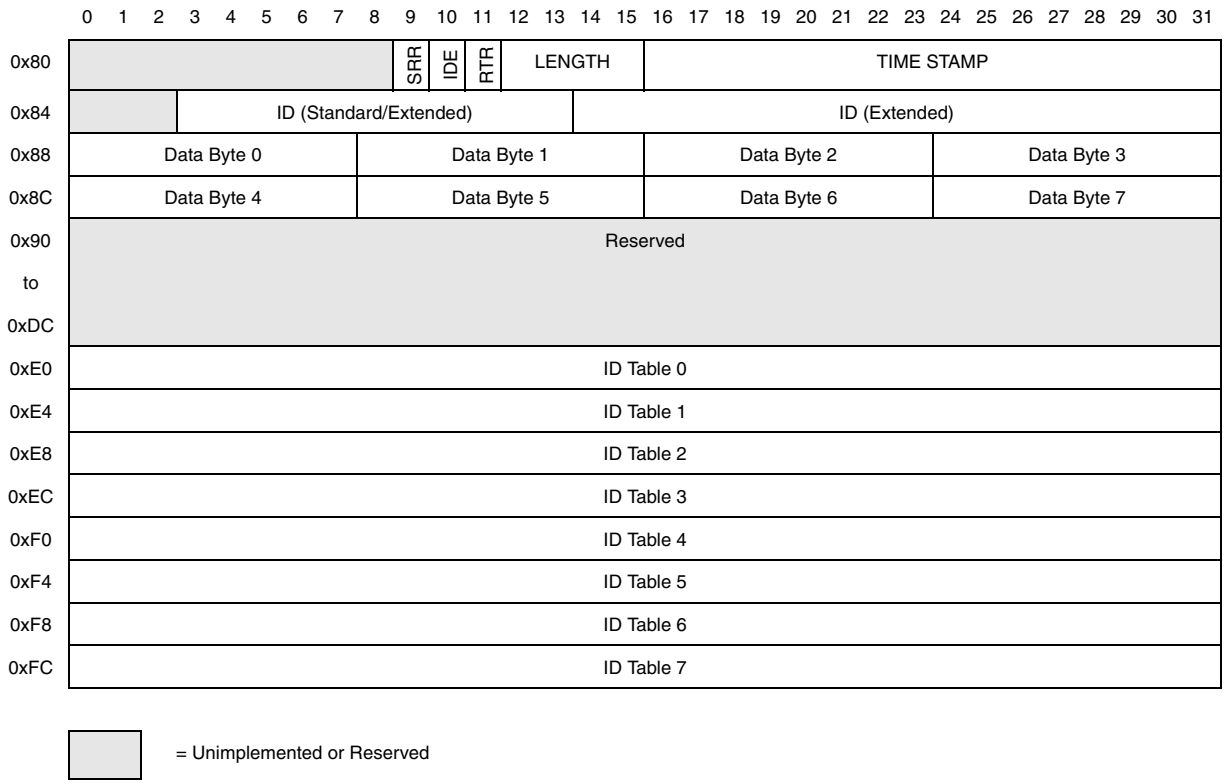


Figure 205. ID Table 0 – 7

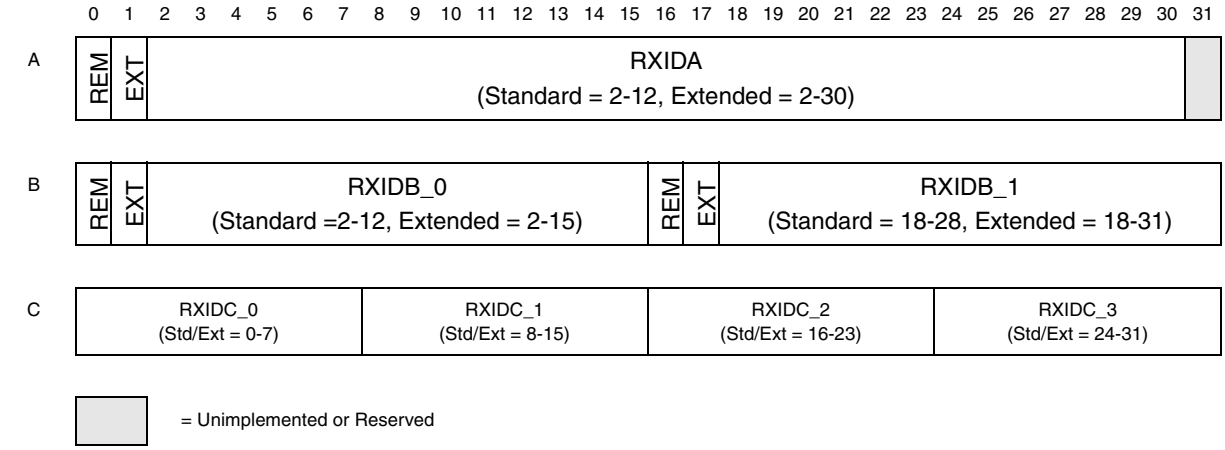


Table 207. Rx FIFO Structure field description

Field	Description
REM	**Remote Frame** This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 1 = Remote Frames can be accepted and data frames are rejected 0 = Remote Frames are rejected and data frames can be accepted
EXT	**Extended Frame** Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 1 = Extended frames can be accepted and standard frames are rejected 0 = Extended frames are rejected and standard frames can be accepted
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.

22.3.4 Register 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. This register can be accessed at any time, however some fields must be changed only during Freeze Mode. Find more information in the fields descriptions ahead.

Figure 206. Module Configuration Register (MCR)

Offset: 0x0000 Access: 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	Note (1)	1	0	1	1	0	0	Note (2)	1	0	0	Note (3)	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

1. Reset value of this bit is different on various platforms. Consult the specific MCU documentation to determine its value.
2. Different on various platforms, but it is always the opposite of the MDIS reset value.
3. Different on various platforms, but it is always the same as the MDIS reset value.

Table 208. MCR field descriptions

Field	Description
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. 1 = Disable the FlexCAN module 0 = Enable the FlexCAN module
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 enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 1 = Enabled to enter Freeze Mode 0 = Not enabled to enter Freeze Mode
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 22.3.3 Rx FIFO structure* and *Section 22.4.8 Rx FIFO* for more information. This bit must be written in Freeze mode only. 1 = FIFO enabled 0 = FIFO not enabled

Table 208. MCR field descriptions (continued)

Field	Description
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 cannot be entered while FlexCAN is in the low power mode. See *Section Freeze Mode* for more information. 1 = Enters Freeze Mode if the FRZ bit is asserted. 0 = No Freeze Mode request.
NOT_RDY	**FlexCAN Not Ready** This read-only bit indicates that FlexCAN is either in Disable Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 1 = FlexCAN module is either in Disable Mode or Freeze Mode 0 = FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode
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, IMASK2, IFLAG1, IFLAG2. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: – CTRL – RXIMR0–RXIMR63 – RXGMASK, RX14MASK, RX15MASK – all Message Buffers The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR, but it is also asserted when global soft reset is requested at MCU level. 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 the low power mode. The module should be first removed from low power mode, and then soft reset can be applied. 1 = Resets the registers marked as “affected by soft reset” in *Table 202* 0 = No reset request
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 the low power mode, then the FRZ_ACK bit will only be set when the low power mode is exited. See *Section Freeze Mode* for more information. 1 = FlexCAN in Freeze Mode, prescaler stopped 0 = FlexCAN not in Freeze Mode, prescaler running

Table 208. MCR field descriptions (continued)

Field	Description
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 202*. Reset value of this bit is '1', so the affected registers start with Supervisor access restrictions.This bit should be written in Freeze mode only. 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 0 = Affected registers are in Unrestricted memory space
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. This bit must be written in Freeze mode only. 1 = TWRN_INT and RWRN_INT bits are set when the respective error counter transition from < 96 to ≥ 96. 0 = TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters.
LPM_ACK	**Low Power Mode Acknowledge** This read-only bit indicates that FlexCAN is in Disable Mode. This low power mode cannot 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* for more information. 1 = FlexCAN is in Disable Mode. 0 = FlexCAN is not in Disable Mode
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. This bit must be written in Freeze mode only. 1 = Self reception disabled 0 = Self reception enabled
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. This bit must be written in Freeze mode only. 1 = Individual Rx masking and queue feature are enabled. 0 = Individual Rx masking and queue feature are disabled.

Table 208. MCR field descriptions (continued)

Field	Description
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 is used to extend 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. This bit must be written in Freeze mode only. 1 = Local Priority enabled 0 = Local Priority disabled
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. This bit must be written in Freeze mode only. 1 = Abort enabled 0 = Abort disabled
IDAM	**ID Acceptance Mode** This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 209*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 22.3.3 Rx FIFO structure*. This bit must be written in Freeze mode only.
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 must be changed only while the module is in Freeze Mode. **Maximum MBs in use = MAXMB + 1** *Note: MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages.*

Table 209. 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. This register can be accessed at any time, however some fields must be changed only during either Disable Mode or Freeze Mode. Find more information in the fields descriptions ahead.

Figure 207. Control Register (CTRL)

Offset: 0x0004 Access: 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 210. CTRL field descriptions

Field	Description
PRESDIV	**Prescaler Division Factor** This 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*. This bit must be written in Freeze mode only. **Sclock frequency = CPI clock frequency / (PRESDIV + 1)**
RJW	**Resync Jump Width** This 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. This bit must be written in Freeze mode only. **Resync Jump Width = RJW + 1.**
PSEG1	**PSEG1 — Phase Segment 1** This field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only. **Phase Buffer Segment 1 = (PSEG1 + 1) x Time-Quanta.**
PSEG2	**PSEG2 — Phase Segment 2** This field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. This bit must be written in Freeze mode only. **Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.**
BOFF_MSK	**Bus Off Mask** This bit provides a mask for the Bus Off Interrupt. 1= Bus Off interrupt enabled 0 = Bus Off interrupt disabled

Table 210. CTRL field descriptions (continued)

Field	Description
ERR_MSK	**Error Mask** This bit provides a mask for the Error Interrupt. 1 = Error interrupt enabled 0 = Error interrupt disabled
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 FMPLL) 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. 1 = The CAN engine clock source is the bus clock 0 = The CAN engine clock source is the oscillator clock *Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a FMPLL, in which case it would have only the oscillator clock, or it may use only the FMPLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation.*
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. This bit must be written in Freeze mode only. 1 = Loop Back enabled 0 = Loop Back disabled
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. 1 = Tx Warning Interrupt enabled 0 = Tx Warning Interrupt disabled
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. 1 = Rx Warning Interrupt enabled 0 = Rx Warning Interrupt disabled
SMP	**Sampling Mode** This bit defines the sampling mode of CAN bits at the Rx input. This bit must be written in Freeze mode only. 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 0 = Just one sample is used to determine the bit value

Table 210. CTRL field descriptions (continued)

Field	Description
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. 1 = Automatic recovering from Bus Off state disabled 0 = Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B
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. This bit must be written in Freeze mode only. 1 = Timer Sync feature enabled 0 = Timer Sync feature disabled
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. This bit must be written in Freeze mode only. 1 = Lowest number buffer is transmitted first 0 = Buffer with highest priority is transmitted first
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. 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. This bit must be written in Freeze mode only. 1 = FlexCAN module operates in Listen-Only Mode 0 = Listen-Only Mode is deactivated
PROPSEG	**Propagation Segment** This field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only. 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 208. Free Running Timer (TIMER)

Offset: 0x0008 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	TIMER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Rx Global Mask (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 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 acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in 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.

Refer to *Section 22.4.8 Rx FIFO* for important details on usage of RXGMASK on filtering process for Rx FIFO.

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 209. Rx Global Mask Register (RXGMASK)

Offset: 0x0010 Access: 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 211. RXGMASK field description

Field	Description
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). 1 = The corresponding bit in the filter is checked against the one received 0 = the corresponding bit in the filter is “don’t care”

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 MCR causes the RX14MASK Register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

Refer to *Section 22.4.8 Rx FIFO* for important details on usage of RX14MASK on filtering process for Rx FIFO.

It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x14
- Reset Value: 0xFFFF_FFFF

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 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 MCR is set (FIFO enabled), the RXG14MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

See *Section 22.4.8 Rx FIFO* for important details on usage of RXG15MASK on filtering process for Rx FIFO.

It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x18
- Reset Value: 0xFFFF_FFFF

Error Counter Register (ECR)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (TX_ERR_COUNTER field) and 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 in Freeze 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 zero.
- 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 zero 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 210. Error Counter Register (ECR)

Offset: 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	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 16–23. Bits 22–28 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT, WAK_INT and ERR_INT, that are interrupt flags that can be cleared by writing '1' to them (writing '0' has no effect). See *Section 22.4.11 Interrupts* for more details.

Figure 211. Error and Status Register (ESR)

Offset: 0x0020 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	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

= Unimplemented or Reserved

Table 212. ESR field descriptions

Field	Description
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. 1 = The Tx error counter transition from < 96 to ≥ 96 0 = No such occurrence
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. 1 = The Rx error counter transition from < 96 to ≥ 96 0 = No such occurrence
BIT1_ERR	**Bit1 Error** This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 1 = At least one bit sent as recessive is received as dominant 0 = No such occurrence *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.*

Table 212. ESR field descriptions (continued)

Field	Description
BIT0_ERR	**Bit0 Error** This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 1 = At least one bit sent as dominant is received as recessive 0 = No such occurrence
ACK_ERR	**Acknowledge Error** This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT. 1 = An ACK error occurred since last read of this register 0 = No such occurrence
CRC_ERR	**Cyclic Redundancy Check Error** This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received. 1 = A CRC error occurred since last read of this register. 0 = No such occurrence
FRM_ERR	**Form Error** This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit. 1 = A Form Error occurred since last read of this register 0 = No such occurrence
STF_ERR	**Stuffing Error** This bit indicates that a Stuffing Error has been detected. 1 = A Stuffing Error occurred since last read of this register. 0 = No such occurrence.
TX_WRN	**TX Error Warning** This bit indicates when repetitive errors are occurring during message transmission. 1 = TX_Err_Counter ≥ 96 0 = No such occurrence
RX_WRN	**Rx Error Warning** This bit indicates when repetitive errors are occurring during message reception. 1 = Rx_Err_Counter ≥ 96 0 = No such occurrence
IDLE	**CAN bus IDLE state** This bit indicates when CAN bus is in IDLE state. 1 = CAN bus is now IDLE 0 = No such occurrence
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. 1 = FlexCAN is transmitting a message (IDLE = 0) 0 = FlexCAN is receiving a message (IDLE = 0)

Table 212. ESR field descriptions (continued)

Field	Description
FLT_CONF	**Fault Confinement State** This field indicates the Confinement State of the FlexCAN module, as shown in *Table 213*. 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.
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. 1 = FlexCAN module entered ‘Bus Off’ state 0 = No such occurrence
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. 1 = Indicates setting of any Error Bit in the Error and Status Register 0 = No such occurrence

Table 213. Fault confinement state

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

Interrupt Masks 2 Register (IMASK2)

This register allows any number of a range of 32 Message Buffer Interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e. when the corresponding IFLAG2 bit is set).

Figure 212. Interrupt Masks 2 Register (IMASK2)

Offset: 0x0024 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63M	BUF 62M	BUF 61M	BUF 60M	BUF 59M	BUF 58M	BUF 57M	BUF 56M	BUF 55M	BUF 54M	BUF 53M	BUF 52M	BUF 51M	BUF 50M	BUF 49M	BUF 48M
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 47M	BUF 46M	BUF 45M	BUF 44M	BUF 43M	BUF 42M	BUF 41M	BUF 40M	BUF 39M	BUF 38M	BUF 37M	BUF 36M	BUF 35M	BUF 34M	BUF 33M	BUF 32M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 214. IMASK2 field descriptions

Field	Description
BUF63M – BUF32M	**Buffer MB_i Mask** Each bit enables or disables the respective FlexCAN Message Buffer (MB32 to MB63) Interrupt. 1 = The corresponding buffer Interrupt is enabled 0 = The corresponding buffer Interrupt is disabled *Note: Setting or clearing a bit in the IMASK2 Register can assert or negate an interrupt request, if the corresponding IFLAG2 bit is set.*

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 (i.e., when the corresponding IFLAG1 bit is set).

Figure 213. Interrupt Masks 1 Register (IMASK1)

Offset: 0x0028 Access: 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 215. IMASK1 field descriptions

Field	Description
BUF31M – BUF0M	**BUF31M–BUF0M — Buffer MB_i Mask** Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 1 = The corresponding buffer Interrupt is enabled 0 = The corresponding buffer Interrupt is disabled *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 2 Register (IFLAG2)

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG2 bit. If the corresponding IMASK2 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 IFLAG2 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

Figure 214. Interrupt Flags 2 Register (IFLAG2)

Offset: 0x002C Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63I	BUF 62I	BUF 61I	BUF 60I	BUF 59I	BUF 58I	BUF 57I	BUF 56I	BUF 55I	BUF 54I	BUF 53I	BUF 52I	BUF 51I	BUF 50I	BUF 49I	BUF 48I
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 47I	BUF 46I	BUF 45I	BUF 44I	BUF 43I	BUF 42I	BUF 41I	BUF 40I	BUF 39I	BUF 38I	BUF 37I	BUF 36I	BUF 35I	BUF 34I	BUF 33I	BUF 32I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 216. IFLAG2 field descriptions

Field	Description
BUF32I – BUF63I	**Buffer MB_i Interrupt** Each bit flags the respective FlexCAN Message Buffer (MB32 to MB63) interrupt. 1 = The corresponding buffer has successfully completed transmission or reception 0 = No such occurrence

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 8 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 215. Interrupt Flags 1 Register (IFLAG1)

Offset: 0x0030 Access: 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 217. IFLAG1 field descriptions

Field	Description
BUF31I – BUF8I	**Buffer MB$_i$ Interrupt** Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt. 1 = The corresponding MB has successfully completed transmission or reception 0 = No such occurrence
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). 1 = MB7 completed transmission/reception or FIFO overflow 0 = No such occurrence
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 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 1 = MB6 completed transmission/reception or FIFO almost full 0 = No such occurrence
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. 1 = MB5 completed transmission/reception or frames available in the FIFO 0 = No such occurrence
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. 1 = Corresponding MB completed transmission/reception 0 = No such occurrence

Rx Individual Mask Registers (RXIMR0–RXIMR63)

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 SRAM, 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.

Figure 216. Rx Individual Mask Registers (RXIMR0 – RXIMR63)

Base + 0x0004

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0

Table 218. RXIMR0 – RXIMR63 field description

Field	Description
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). 1 = The corresponding bit in the filter is checked against the one received 0 = the corresponding bit in the filter is “don’t care”

22.4 Functional description

22.4.1 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 up to 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 22.3.2 Message buffer structure*). The memory corresponding to the first 8 MBs 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 (up to 8 extended IDs or 16 standard IDs or 32 8-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 205*). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to *Table 206*). 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*).

22.4.2 Local priority transmission

The term local priority refers to the priority of transmit messages of the host node. This allows increased control over the priority mechanism for transmitting messages. *Table 203* shows the placement of PRIO in the ID part of the message buffer.

An additional 3-bit field (PRIO) in the long-word ID part of the message buffer structure has been added for local priority determination. They are prefixed to the regular ID to define the transmission priority. These bits are not transmitted and are intended only for Tx buffers.

Perform the following to use the local priority feature:

1. Set the LPRIO_EN bit in the CANx_MCR.
2. Write the additional PRIO bits in the ID long-word of Tx message buffers when configuring the Tx buffers.

With this extended ID concept, the arbitration process is based on the full 32-bit word. However, the actual transmitted ID continues to have 11 bits for standard frames and 29 bits for extended frames.

22.4.3 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

- 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*).
- Write the ID word.
- Write the data bytes.
- 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 four (see *Table 205* and *Table 206* in *Section 22.3.2 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.

22.4.4 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[s] 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 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 up to eight data bytes, even if the DLC (Data Length Code) value is bigger.

22.4.5 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:

- 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

s. 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.

- Write the ID word
- 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 205* and *Table 206* in *Section 22.3.2 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:

- Read the Control and Status word (mandatory – activates an internal lock for this buffer)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- 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 22.4.7 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 205*. If the CPU tries to workaround 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 do polling by reading directly 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 22.4.8 Rx FIFO*). Upon receiving the frames

available interrupt from FIFO, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

22.4.6 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 cannot 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 205* and *Table 206*). 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 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–RXIMR63)*. 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 SRAM, 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.

22.4.7 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Transmit process and *Section 22.4.5 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 trasmission 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:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- 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 cannot 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 which have a code different than INACTIVE ('0000') or EMPTY[t] ('0100'). Also, Tx MBs cannot 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 cannot 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.

Note: The FlexCAN module has 2 SMBs thus if a message with another ID arrives it is not lost. So overall the probability to lose message is very low unless a series of messages with the same ID arrives, which is not common in FlexCAN.

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.

22.4.8 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 22.3.3 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 up to 6 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 with the

t. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior will be honored when the BCC bit is negated.

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 5 frames 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 22.3.3 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.

22.4.9 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 217 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) submodule. 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 FMPLL). 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).

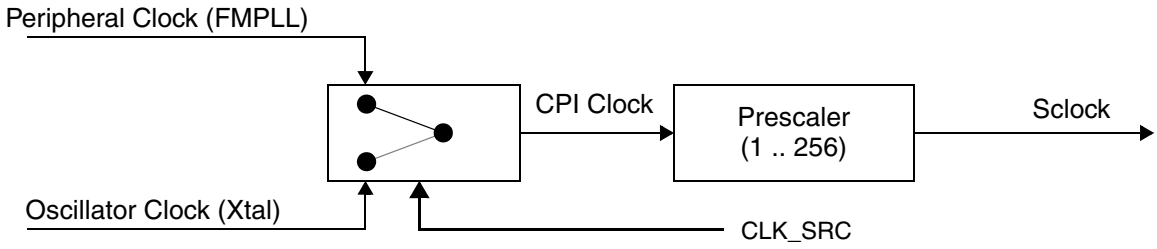


Figure 217. CAN engine clocking scheme

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than FMPLL generated clocks.

Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a FMPLL, in which case it would have only the oscillator clock, or it may use only the FMPLL 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.

$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler\ \ Value)}$$

A bit time is subdivided into three segments[u] (reference *Figure 218* and *Table 219*):

- 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

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

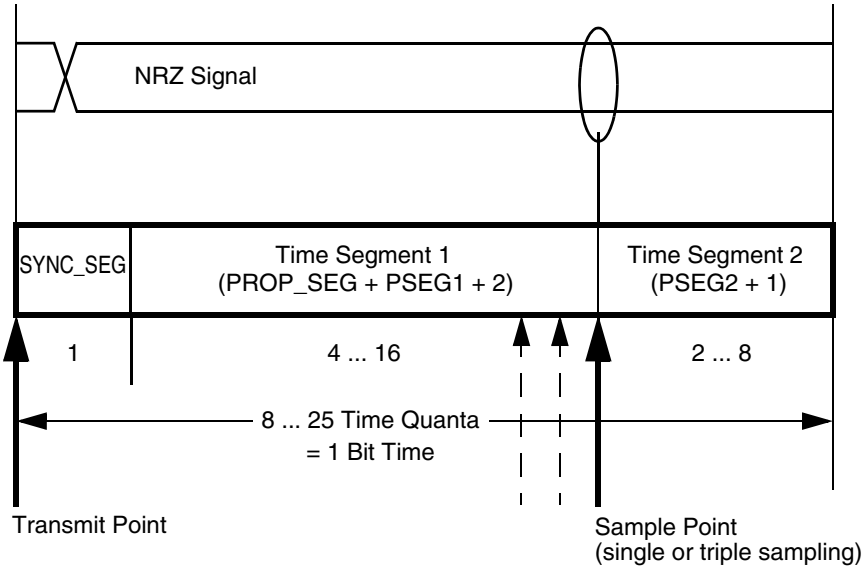


Figure 218. Segments within the bit time

u. 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 219. 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 220 is an example of the CAN compliant segment settings and the related parameter values. It refers to the official CAN specification.

Table 220. 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 219*.

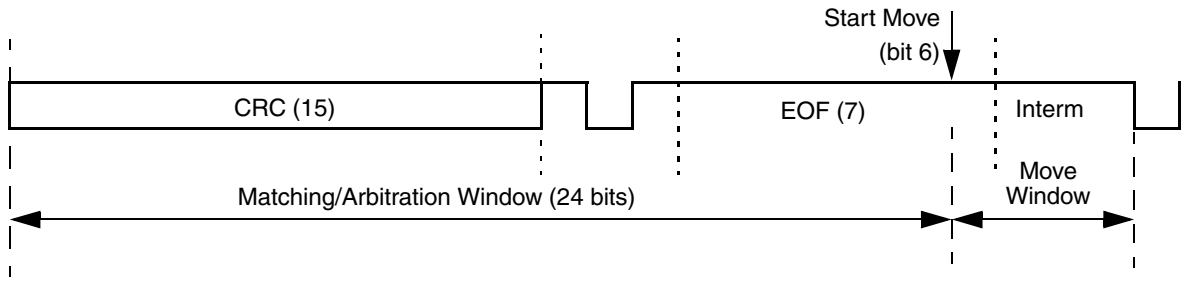


Figure 219. Arbitration, match and move time windows

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 220*
- The peripheral clock frequency cannot be smaller than the oscillator clock frequency, i.e. the FMPLL cannot 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 221*

Table 221. Minimum ratio between peripheral clock frequency and CAN bit rate

Number of message buffers	Minimum ratio
16	8
32	8
64	16

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 221* 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 64 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 one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

22.4.10 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 low power mode (Disable Mode). 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 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 submodules, 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 submodules
- 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.

22.4.11 Interrupts

The module can generate up to 69 interrupt sources (64 interrupts due to message buffers and 5 interrupts due to Ored interrupts from MBs, Bus Off, Error, Tx Warning and Rx Warning). 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 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 which 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 4 interrupt sources (Bus Off, Error, Tx Warning and Rx Warning) generate interrupts like the MB ones, and can be read from the Error and Status Register. The Bus Off, Error, Tx Warning and Rx Warning interrupt mask bits are located in the Control Register, and the Wake-Up interrupt mask bit is located in the MCR.

22.4.12 Bus interface

The CPU access to FlexCAN registers are 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 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 SRAM 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 SRAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. 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 0x047F. The available memory in the Mask Registers space would be from 0x0884 to 0x097F.

Note: Unused MB space must not be used as general purpose SRAM while FlexCAN is transmitting and receiving CAN frames.

22.5 Initialization/Application information

This section provide instructions for initializing the FlexCAN module.

22.5.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to *Table 202* to see what registers are affected by soft reset)
- SOFT_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset cannot be applied while clocks are shut down in the low power mode. The low power mode should be exited and the clocks resumed before applying 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 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) and in CTRL Register (for Bus Off and Error interrupts)
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

22.5.2 FlexCAN addressing and SRAM size configurations

There are three SRAM configurations that can be implemented within the FlexCAN module. The possible configurations are:

- For 16 MBs: 288 bytes for MB memory and 64 bytes for Individual Mask Registers
- For 32 MBs: 544 bytes for MB memory and 128 bytes for Individual Mask Registers
- For 64 MBs: 1056 bytes for MB memory and 256 bytes for Individual Mask Registers

In each configuration the user can program the maximum number of MBs that will take part in the matching and arbitration processes using the MAXMB field in the MCR. For 16 MB configuration, MAXMB can be any number between 0–15. For 32 MB configuration, MAXMB can be any number between 0–31. For 64 MB configuration, MAXMB can be any number between 0–63.

23 Deserial Serial Peripheral Interface (DSPI)

23.1 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 SPC560Bx and SPC560Cx has three identical DSPI modules (DSPI_0, DSPI_1 and DSPI_2). 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.

A block diagram of the DSPI is shown in *Figure 220*.

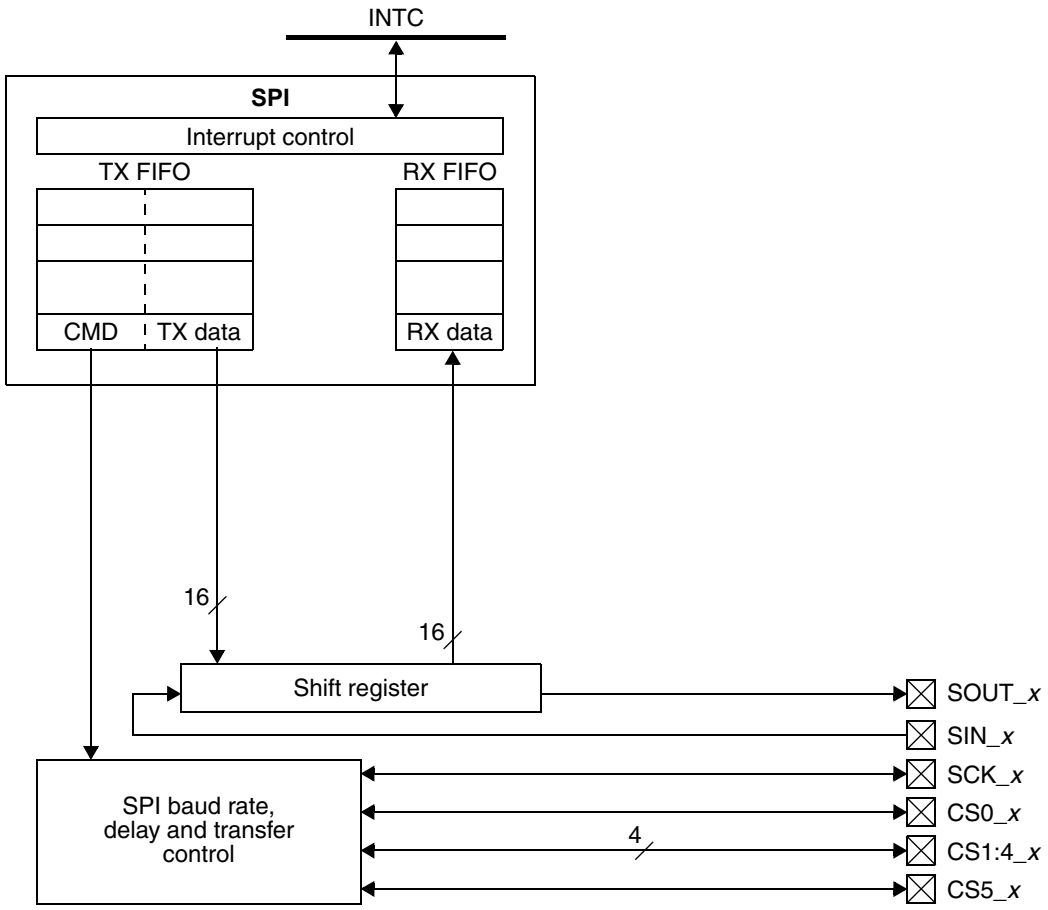


Figure 220. DSPI block diagram

The register content is transmitted using an SPI protocol.

For queued operations the SPI queues reside in internal SRAM which is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through host software.

Figure 221 shows a DSPI with external queues in internal SRAM.

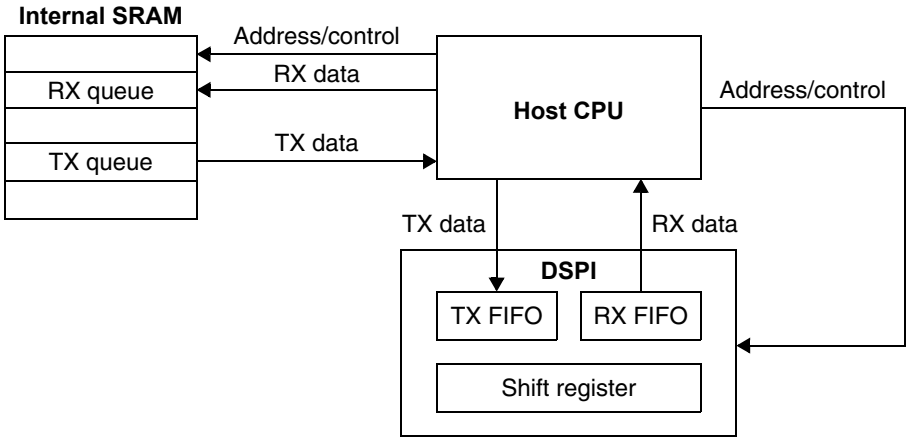


Figure 221. DSPI with queues

23.2 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of four 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
  - 6 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

- Up to 6 peripheral chip selects, expandable to 64 with external demultiplexer
- Deglitching support for up to 32 peripheral chip selects with external demultiplexer
- 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) or (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (SCK)

23.3 Modes of operation

The DSPI has four modes of operation. These modes can be divided into two categories:

- Module-specific: Master, Slave, and Module Disable modes
- MCU-specific: Debug mode

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. An 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.

23.3.1 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, see *Section , Master mode*.

23.3.2 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, see *Section , Slave mode*.

23.3.3 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, see *Section , Module Disable mode*.

23.3.4 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 and remains dictated by the module-specific mode and configuration of the DSPI.

For more information, see *Section , Debug mode*.

23.4 External signal description

23.4.1 Signal overview

Table 222 lists off-chip DSPI signals.

Table 222. Signal properties

Name	I/O type	Function	
		Master mode	Slave mode
CS0_*x*	Output / input	Peripheral chip select 0	Slave select
CS1:3_x	Output	Peripheral chip select 1–3	Unused(1)
CS4_x	Output	Peripheral chip select 4	Master trigger
CS5_x	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.

23.4.2 Signal names and descriptions

Peripheral Chip Select / Slave Select (CS0_x)

In master mode, the CS0_x signal is a peripheral chip select output that selects the slave device to which the current transmission is intended.

In slave mode, the CS0_x signal is a slave select input signal that allows an SPI master to select the DSPI as the target for transmission. 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]).

Set the IBE and OBE bits in the SIU_PCR for all CS0_x 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.

Peripheral Chip Selects 1–3 (CS1:3_x)

CS1:3_x are peripheral chip select output signals in master mode. In slave mode these signals are not used.

Peripheral Chip Select 4 (CS4_x)

CS4_x is a peripheral chip select output signal in master mode.

Peripheral Chip Select 5 / Peripheral Chip Select Strobe (CS5_x)

CS5_x 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 CS5_x signal is used to select the slave device that receives the current transfer.

CS5_x 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 CS5_x signal indicates the timing to decode CS0:4_x signals, which prevents glitches from occurring.

CS5_x 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.

23.5 Memory map and register description

23.5.1 Memory map

Table 223 shows the DSPI memory map.

Table 223. DSPI memory map

Base addresses: 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2)		
Address offset	**Register**	**Location**
0x00	DSPI Module Configuration Register (DSPI*x*_MCR)	*on page 23-466*
0x04	Reserved	
0x08	DSPI Transfer Count Register (DSPI*x*_TCR)	*on page 23-469*

Table 223. DSPI memory map (continued)

Base addresses: 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2)		
Address offset	**Register**	**Location**
0x0C	DSPI Clock and Transfer Attributes Register 0 (DSPI*x*_CTAR0)	*on page 23-470*
0x10	DSPI Clock and Transfer Attributes Register 1 (DSPI*x*_CTAR1)	*on page 23-470*
0x14	DSPI Clock and Transfer Attributes Register 2 (DSPI*x*_CTAR2)	*on page 23-470*
0x18	DSPI Clock and Transfer Attributes Register 3 (DSPI*x*_CTAR3)	*on page 23-470*
0x1C	DSPI Clock and Transfer Attributes Register 4 (DSPI*x*_CTAR4)	*on page 23-470*
0x20	DSPI Clock and Transfer Attributes Register 5 (DSPI*x*_CTAR5)	*on page 23-470*
0x24–0x28	Reserved	
0x2C	DSPI Status Register (DSPI*x*_SR)	*on page 23-478*
0x30	DSPI Interrupt Request Enable Register (DSPIx_RSER)	*on page 23-480*
0x34	DSPI Push TX FIFO Register (DSPI*x*_PUSHR)	*on page 23-482*
0x38	DSPI Pop RX FIFO Register (DSPI*x*_POPR)	*on page 23-484*
0x3C	DSPI Transmit FIFO Register 0 (DSPI*x*_TXFR0)	*on page 23-485*
0x40	DSPI Transmit FIFO Register 1 (DSPI*x*_TXFR1)	*on page 23-485*
0x44	DSPI Transmit FIFO Register 2 (DSPI*x*_TXFR2)	*on page 23-485*
0x48	DSPI Transmit FIFO Register 3 (DSPI*x*_TXFR3)	*on page 23-485*
0x4C–0x78	Reserved	
0x7C	DSPI Receive FIFO Register 0 (DSPI*x*_RXFR0)	*on page 23-485*
0x80	DSPI Receive FIFO Register 1 (DSPI*x*_RXFR1)	*on page 23-485*
0x84	DSPI Receive FIFO Register 2 (DSPI*x*_RXFR2)	*on page 23-485*
0x88	DSPI Receive FIFO Register 3 (DSPI*x*_RXFR3)	*on page 23-485*

23.5.2 DSPI Module Configuration Register (DSPI*x*_MCR)

The DSPI*x*_MCR contains bits which 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 222. DSPI Module Configuration Register (DSPI*x*_MCR)

Offset: 0x00 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	0	0	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	0	0	0	0	0	0	HALT
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 224. DSPI*x*_MCR field descriptions

Field	Description
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
CONT_SCKE	Continuous SCK enable Enables the serial communication clock (SCK) to run continuously. See *Section 23.6.6, Continuous serial communications clock*, for details. 0 Continuous SCK disabled 1 Continuous SCK enabled *Note: If the FIFO is enabled with continuous SCK mode, the TX-FIFO should be cleared before setting the CONT_SCKE bit, and only the CTAR0 register should be used to transfer attributes; otherwise, a change in SCK frequency occurs.*
DCONF	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

Table 224. DSPI*x*_MCR field descriptions (continued)

Field	Description
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
MTFE	Modified timing format enable Enables a modified transfer format to be used. See *Section , Modified SPI transfer format (MTFE = 1, CPHA = 1)*, for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
PCSSE	Peripheral chip select strobe enable Enables the CS5_*x* to operate as a CS strobe output signal. See *Section , Peripheral chip select strobe enable (CS5_x)*, for more information. 0 CS5_*x* is used as the Peripheral chip select 5 signal 1 CS5_*x* as an active-low CS strobe signal
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. See *Section , Receive FIFO Overflow Interrupt Request (RFOF)*, for more information. 0 Incoming data is ignored 1 Incoming data is put in the shift register
PCSIS*n*	Peripheral chip select inactive state Determines the inactive state of the CS0_*x* signal. CS0_*x* must be configured as inactive high for slave mode operation. 0 The inactive state of CS0_*x* is low 1 The inactive state of CS0_*x* is high
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. See *Section 23.6.8, Power saving features* for more information. 0 Enable DSPI clocks 1 Allow external logic to disable DSPI clocks

Table 224. DSPI*x*_MCR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>DIS_TXF</td><td>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. See Section , FIFO disable operation for details.

0 TX FIFO is enabled
1 TX FIFO is disabled</td></tr>
<tr><td>DIS_RXF</td><td>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. See Section , FIFO disable operation for details.
0 RX FIFO is enabled
1 RX FIFO is disabled</td></tr>
<tr><td>CLR_TXF</td><td>Clear TX FIFO. CLR_TXF is used to flush the TX FIFO. Writing a ‘1’ to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero.
0 Do not clear the TX FIFO Counter
1 Clear the TX FIFO Counter</td></tr>
<tr><td>CLR_RXF</td><td>Clear RX FIFO. CLR_RXF is used to flush the RX FIFO. Writing a ‘1’ to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero.
0 Do not clear the RX FIFO Counter
1 Clear the RX FIFO Counter</td></tr>
<tr><td>SMPL_PT</td><td>Sample point
Allows the host software to select when the DSPI master samples SIN in modified transfer format. Figure 237 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>Reserved</td></tr></table></td></tr>
<tr><td>HALT</td><td>Halt
Provides a mechanism for software to start and stop DSPI transfers. See Section 23.6.2, Start and stop of DSPI transfers, for details on the operation of this bit.
0 Start transfers
1 Stop transfers</td></tr>
</table>

23.5.3 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 223. DSPI Transfer Count Register (DSPIx_TCR)

Offset: 0x08 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SPI_TCNT															
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 225. DSPI*x*_TCR field descriptions

Field	Description
SPI_TCNT	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.

23.5.4 DSPI Clock and Transfer Attributes Registers 0–5 (DSPI*x*_CTAR*n*)

The DSPI modules each contain six clock and transfer attribute registers (DSPI*x*_CTAR*n*) which are used to 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 in the SPC560Bx and SPC560Cx 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 are used to set the slave transfer attributes. See 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 224. DSPI Clock and Transfer Attributes Registers 0–5 (DSPI*x*_CTARn)

Offsets: 0x0C–0x20 (6 registers) Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSBFE	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 226. DSPI*x*_CTAR*n* field descriptions

Field	Descriptions
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 233*. 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
FMSZ	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 234* lists the frame size encodings.
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
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 226. DSPI*x*_CTAR*n* field descriptions (continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>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>PCSSCK</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 below lists the prescaler values. See the CSSCK 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>PASC</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 below 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>PDT</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 below 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 226. DSPI*x*_CTAR*n* field descriptions (continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>PBR</td><td>Baud Rate Prescaler
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 below. 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>CSSCK</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 235 list 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 8 $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$
See Section , CS to SCK delay (tCSC) for more details.</td></tr>
<tr><td>ASC</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 236 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 9 $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$
See Section , After SCK delay (tASC) for more details.</td></tr>
<tr><td>DT</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 237 lists the scaler values. In the 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 10 $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$
See Section , Delay after transfer (tDT) for more details.</td></tr>
</table>

Table 226. DSPI*x*_CTAR*n* field descriptions (continued)

Field	Descriptions
BR	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. *Table 238* lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: **Equation 11** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section , CS to SCK delay (tCSC)* for more details.

Table 227. 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 228. 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 229. 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

Table 229. DSPI PCS to SCK delay scaler (continued)

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to SCK delay scaler value
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 230. 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 231. 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 232. 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

Table 232. DSPI baud rate scaler (continued)

BR	Baud rate scaler value	BR	Baud rate scaler value
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

Table 233. 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 234. 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 235. 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

Table 235. DSPI PCS to SCK delay scaler (continued)

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to SCK delay scaler value
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 236. 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 237. 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 238. 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

Table 238. DSPI baud rate scaler (continued)

BR	Baud rate scaler value	BR	Baud rate scaler value
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

23.5.5 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 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. This register may not be writable in Module Disable mode due to the use of power saving mechanisms.

Figure 225. DSPI Status Register (DSPI*x*_SR)

Offset: 0x2C Access: R/W

	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	1	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 239. DSPI*x*_SR field descriptions

Field	Description
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 t_{ASC} delay starts. See *Section , Classic SPI transfer format (CPHA = 0)* for details. 0 Transfer not complete 1 Transfer complete
TXRXS	TX and RX status Reflects the status of the DSPI. See *Section 23.6.2, 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 239. DSPI*x*_SR field descriptions (continued)

Field	Description
EOQF	End of queue flag Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. See *Section , Classic SPI transfer format (CPHA = 0)* for details. 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 *Note: EOQF does not function in slave mode.*
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. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
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 Edam controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full
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. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
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 Edam controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty *Note: In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.*
TXCTR	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.

Table 239. DSPI*x*_SR field descriptions (continued)

Field	Description
TXNXTPTR	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. See *Section , Transmit First In First Out (TX FIFO) buffering mechanism* for more details.
RXCTR	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 t_{ASC} delay starts. See *Section , Classic SPI transfer format (CPHA = 0)* for details.
POPNXTPTR	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. See *Section , Receive First In First Out (RX FIFO) buffering mechanism* for more details.

23.5.6 DSPI Interrupt Request Enable Register (DSPI*x*_RSER)

The DSPI*x*_RSER enables flag bits in the DSPI*x*_SR to generate interrupt requests.

Do not write to the DSPI*x*_RSER while the DSPI is running.

Figure 226. DSPI Interrupt Request Enable Register (DSPIx_RSER)

Offset:0x30 Access: 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 240. DSPIx_RSER field descriptions

Field	Description
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
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
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
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 an interrupt request. 0 TFFF interrupt requests are disabled 1 TFFF interrupt requests are enabled
TFFF_DIRS	Transmit FIFO fill interrupt request select Selects 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 an interrupt request. 0 Interrupt request is selected 1 Reserved
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
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 an interrupt request. 0 RFDF interrupt requests are disabled 1 RFDF interrupt requests are enabled
RFDF_DIRS	Receive FIFO drain interrupt request select Selects 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 an interrupt request. 0 Interrupt request is selected 1 Reserved

23.5.7 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. See *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 transfers 32 bits to the TX FIFO.

Note: TXDATA is used in master and slave modes.

Figure 227. DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

Offset:0x34 Access: 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	0	0	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 241. DSPI*x*_PUSHR field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>CONT</td><td>Continuous peripheral chip select enable
Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected CS signals to remain asserted between transfers. See 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</td></tr>
<tr><td>CTAS</td><td>Clock and transfer attributes select
Selects which of the DSPIx_CTARs is used to set the transfer attributes for the SPI frame. In SPI slave mode, DSPIx_CTAR0 is used. The following table shows how the CTAS values map to the DSPIx_CTARs. There are eight DSPIx_CTARs in the device DSPI implementation.
Note: Use in SPI master mode only.

<table>
<tr><th>CTAS</th><th>Use clock and transfer attributes from</th></tr>
<tr><td>000</td><td>DSPIx_CTAR0</td></tr>
<tr><td>001</td><td>DSPIx_CTAR1</td></tr>
<tr><td>010</td><td>DSPIx_CTAR2</td></tr>
<tr><td>011</td><td>DSPIx_CTAR3</td></tr>
<tr><td>100</td><td>DSPIx_CTAR4</td></tr>
<tr><td>101</td><td>DSPIx_CTAR5</td></tr>
<tr><td>110</td><td>Reserved</td></tr>
<tr><td>111</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>EOQ</td><td>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 DSPIx_SR is set.

0 The SPI data is not the last data to transfer
1 The SPI data is the last data to transfer
Note: Use in SPI master mode only.</td></tr>
<tr><td>CTCNT</td><td>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 DSPIx_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.

0 Do not clear SPI_TCNT field in the DSPIx_TCR
1 Clear SPI_TCNT field in the DSPIx_TCR
Note: Use in SPI master mode only.</td></tr>
</table>

Table 241. DSPI*x*_PUSHR field descriptions (continued)

Field	Description
PCS*x*	Peripheral chip select *x* Selects which CS*x* signals are asserted for the transfer. 0 Negate the CS*x* signal 1 Assert the CS*x* signal *Note: Use in SPI master mode only.*
TXDATA	Transmit data Holds SPI data for transfer according to the associated SPI command. *Note: Use TXDATA in master and slave modes.*

23.5.8 DSPI POP RX FIFO Register (DSPI*x*_POPR)

The DSPI*x*_POPR allows you to read the RX FIFO. See *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 entry for DSPIx_POPR as guarded.

Figure 228. DSPI POP RX FIFO Register (DSPI*x*_POPR)

Offset:0x38 Access: 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	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 242. DSPI*x*_POPR field descriptions

Field	Description
RXDATA	Received data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNXTPTR).

23.5.9 DSPI Transmit FIFO Registers 0–3 (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 four registers to implement the TX FIFO, that is DSPI*x*_TXFR0–DSPI*x*_TXFR3 are used.

Figure 229. DSPI Transmit FIFO Register 0–3 (DSPI*x*_TXFR*n*)

Offsets: 0x3C–0x48 (4 registers) Access: Read

	0	1	2	3	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	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 243. DSPI*x*_TXFR*n* field descriptions

Field	Description
TXCMD	Transmit command Contains the command that sets the transfer attributes for the SPI data. See *Section 23.5.7, DSPI PUSH TX FIFO Register (DSPIx_PUSHR)*, for details on the command field.
TXDATA	Transmit data Contains the SPI data to be shifted out.

DSPI Receive FIFO Registers 0–3 (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 four registers to implement the RX FIFO, that is DSPI*x*_RXFR0–DSPI*x*_RXFR3 are used.

Figure 230. DSPI Receive FIFO Registers 0–3 (DSPI*x*_RXFR*n*)

Offsets: 0x7C–0x88 (4 registers) Access: 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	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 244. DSPI*x*_RXFR*n* field description

Field	Description
RXDATA	Receive data Contains the received SPI data.

23.6 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 has one configuration, namely serial peripheral interface (SPI), 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. See *Table 224* for the DSPI configuration values.

The DSPI*x*_CTAR0–DSPI*x*_CTAR5 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 231* illustrates how master and slave data is exchanged.

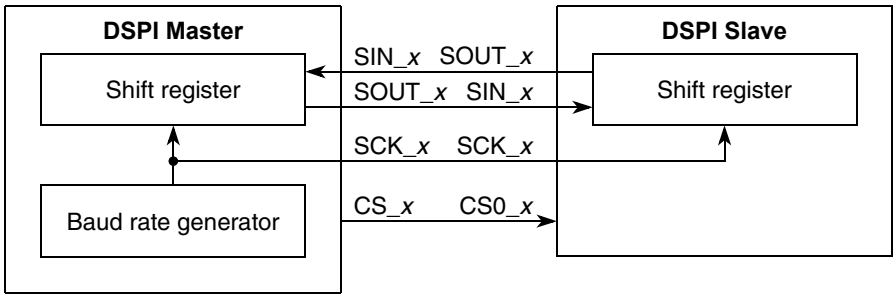


Figure 231. SPI serial protocol overview

The DSPI has six peripheral chip select (CS*x*) signals that are be used to select which of the slaves to communicate with.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in *Section 23.6.5, Transfer formats*. The transfer rate and delay settings are described in *Section 23.6.4, DSPI baud rate and clock delay generation*.

See *Section 23.6.8, Power saving features*, for information on the power-saving features of the DSPI.

23.6.1 Modes of operation

The DSPI modules have the following available distinct modes:

- Master mode
- Slave mode
- Module Disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes whereas debug mode is device-specific.

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 are used to set the transfer attributes. Transfer attribute control is on a frame by frame basis.

See *Section 23.6.3, 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 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 which must be configured in the DSPI slave to communicate correctly.

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.

See *Section 23.6.8, 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.

See *Figure 232* for a state diagram.

23.6.2 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 DSPIx_SR is set in the RUNNING state.

Figure 232 shows a state diagram of the start and stop mechanism.

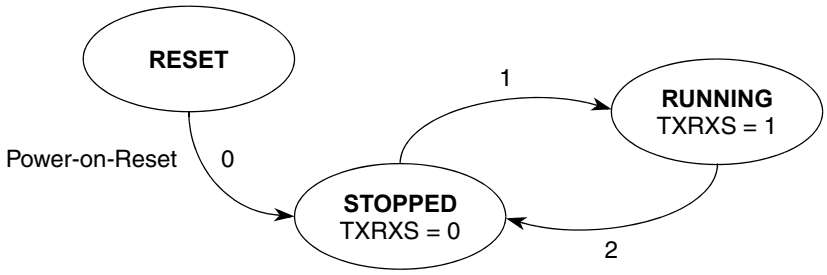


Figure 232. DSPI start and stop state diagram

The transitions are described in *Table 245*.

Table 245. State transitions for start and stop of DSPI transfers

Transition No.	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.

23.6.3 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 SRAM external to the DSPI. Host software 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 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 request conditions are described in *Section 23.6.7, Interrupt 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 (CS*x*) signals. The SPI command field in the executing TX FIFO entry determines which CTARs are used to set the transfer attributes and which CS*x* signal 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.

See *Section 23.5.7, 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 four 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). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO. For more information on DSPIx_PUSHR, see *Section 23.5.7, DSPI PUSH TX FIFO Register (DSPIx_PUSHR)*.

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.

See *Section 23.5.5, 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 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 or alternatively by host software writing a ‘1’ to the TFFF in the DSPI*x*_SR. The TFFF then generates an interrupt request.

See *Section , Transmit FIFO Fill Interrupt 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.

See *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 four 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.

See *Section 23.5.8, 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 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.

See *Section 23.5.8, 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; alternatively the RFDF bit can be cleared by the host writing a ‘1’ to it.

23.6.4 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 233 shows conceptually how the SCK signal is generated.

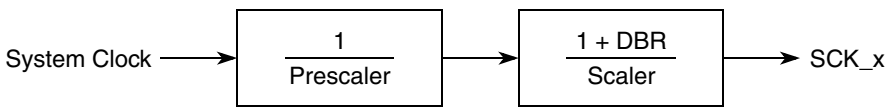


Figure 233. Communications clock prescalers and scalers

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:

$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 246 shows an example of a computed baud rate.

Table 246. Baud rate computation example

f_{SYS}	PBR	Prescaler value	BR	Scaler value	DBR value	Baud rate
64 MHz	0b00	2	0b0000	2	0	16 Mbit/s
20 MHz	0b00	2	0b0000	2	1	10 Mbit/s

CS to SCK delay (t_{CSC})

The CS_*x* to SCK_*x* delay is the length of time from assertion of the CS_*x* signal to the first SCK_*x* edge. See *Figure 235* for an illustration of the CS_*x* to SCK_*x* delay. The PCSSCK and CSSCK fields in the DSPI*x*_CTAR*n* registers select the CS_*x* to SCK_*x* delay, and the relationship is expressed by the following formula:

$$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 247 shows an example of the computed CS to SCK_*x* delay.

Table 247. CS to SCK delay computation example

PCSSCK	Prescaler value	CSSCK	Scaler value	f_{SYS}	CS to SCK delay
0b01	3	0b0100	32	64 MHz	1.5 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 negation of CS_*x*. See *Figure 235* and *Figure 236* 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:

$$t_{ASC} = \frac{1}{f_{SYS}} \times \text{PASC} \times \text{ASC}$$

Table 248 shows an example of the computed after SCK delay.

Table 248. After SCK delay computation example

PASC	Prescaler value	ASC	Scaler value	f_{SYS}	After SCK delay
0b01	3	0b0100	32	64 MHz	1.5 µs

Delay after transfer (t_{DT})

The delay after transfer is the length of time between negation of the CS*x* signal for a frame and the assertion of the CS*x* signal for the next frame. The PDT and DT fields in the DSPI*x*_CTAR*n* registers select the delay after transfer.

See *Figure 235* for an illustration of the delay after transfer.

The following formula expresses the PDT/DT/delay after transfer relationship:

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 249 shows an example of the computed delay after transfer.

Table 249. Delay after transfer computation example

PDT	Prescaler value	DT	Scaler value	f_{SYS}	Delay after transfer
0b01	3	0b1110	32768	64 MHz	1.54 ms

Peripheral chip select strobe enable (CS5_*x*)

The 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 234 shows the timing of the CS5_*x* signal relative to CS signals.

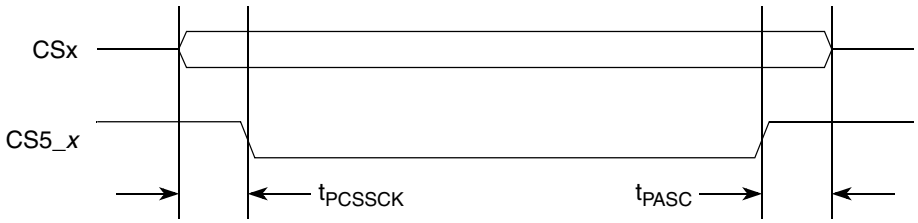


Figure 234. Peripheral chip select strobe timing

The delay between the assertion of the CS*x* signals and the assertion of CS5_*x* is selected by the PCSSCK field in the DSPI*x*_CTAR based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

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:

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 250 shows an example of the computed t_{PCSSCK} delay.

Table 250. Peripheral chip select strobe assert computation example

PCSSCK	Prescaler	f_{SYS}	Delay before transfer
0b11	7	64 MHz	109.4 ns

Table 251 shows an example of the computed the t_{PASC} delay.

Table 251. Peripheral chip select strobe negate computation example

PASC	Prescaler	f_{SYS}	Delay after transfer
0b11	7	64 MHz	109.4 ns

23.6.5 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. See *Section , Continuous selection format* for details.

Classic SPI transfer format (CPHA = 0)

The transfer format shown in *Figure 235* 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.

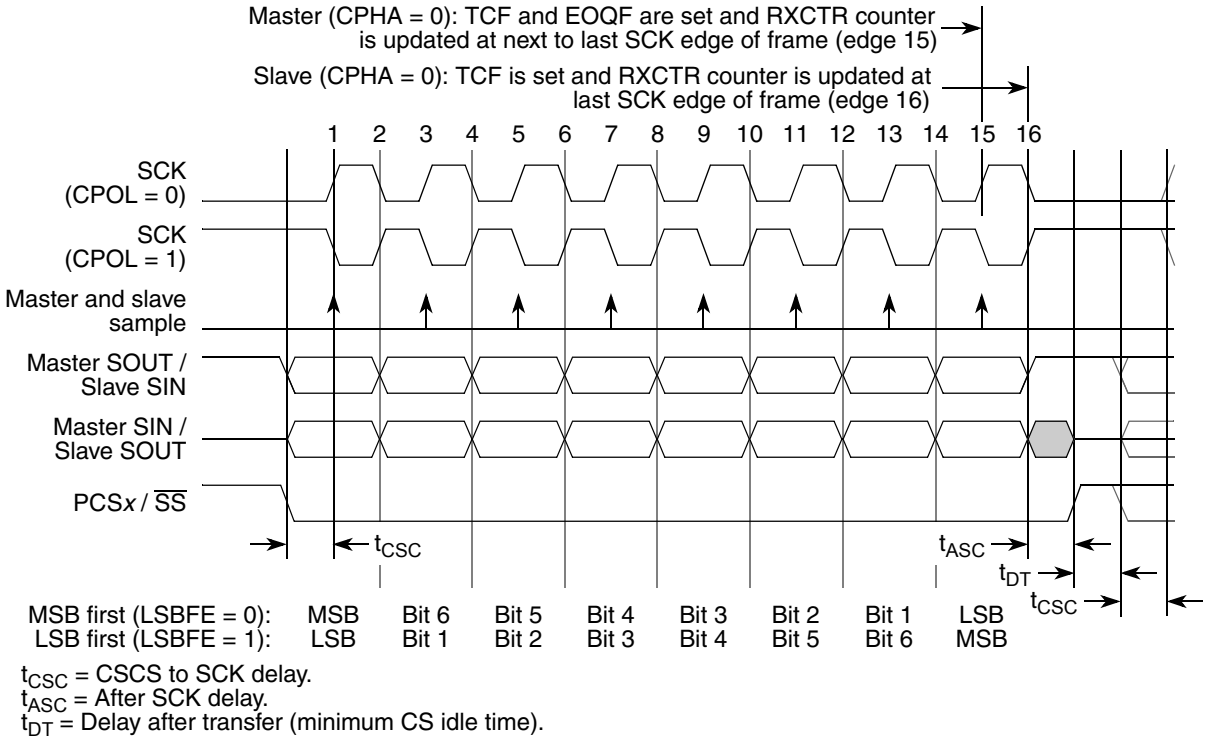


Figure 235. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

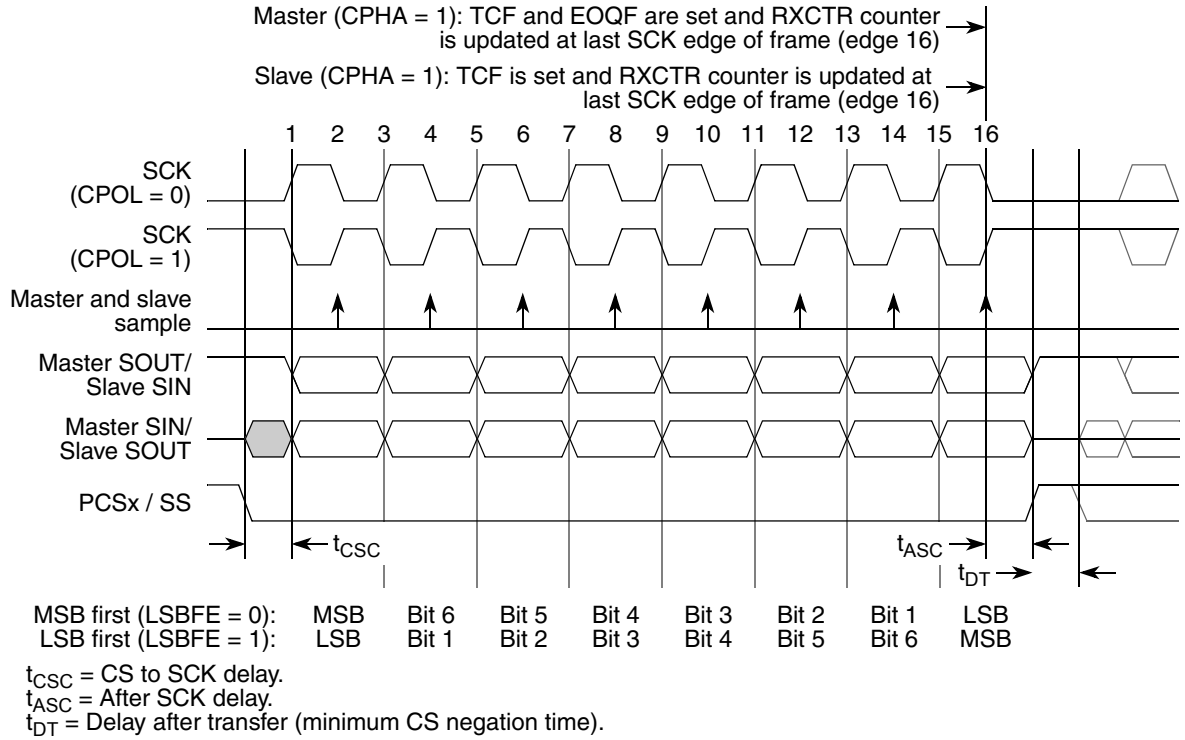
The master initiates the transfer by placing its first data bit on the SOUT_*x* pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT_*x* pin. After the t_{CSC} delay has elapsed, the master outputs the first 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 235*.

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 235*.

Classic SPI transfer format (CPHA = 1)

This transfer format shown in *Figure 236* 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 236. 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 236*. 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 252* 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 252. 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 237 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.

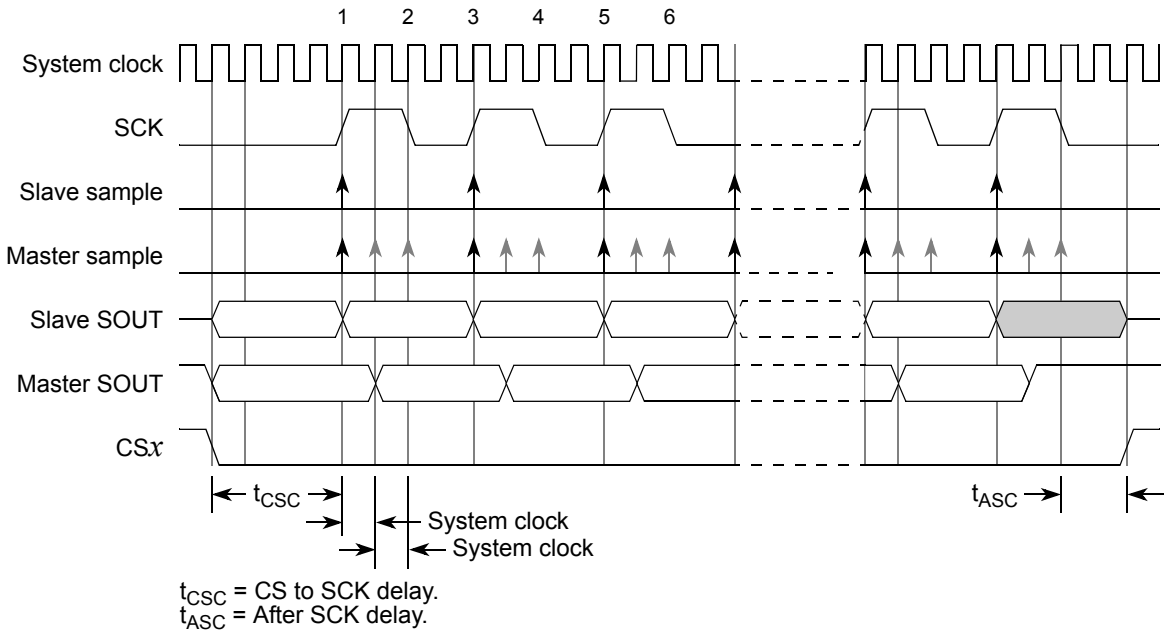


Figure 237. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$)

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 third 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 greater or equal to half of the SCK period.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 238 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is described.

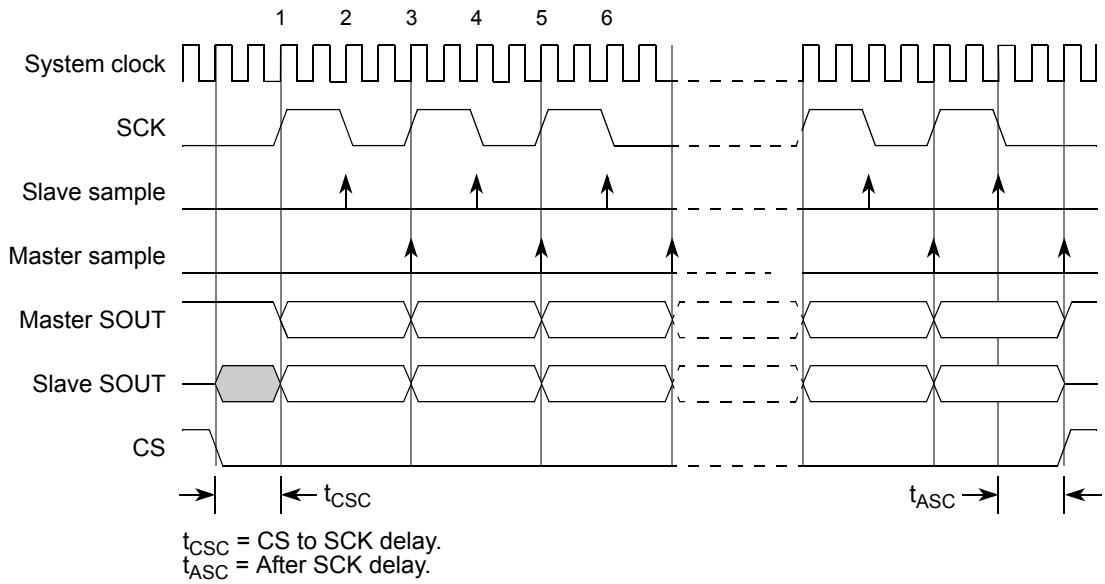


Figure 238. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$)

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 239 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.

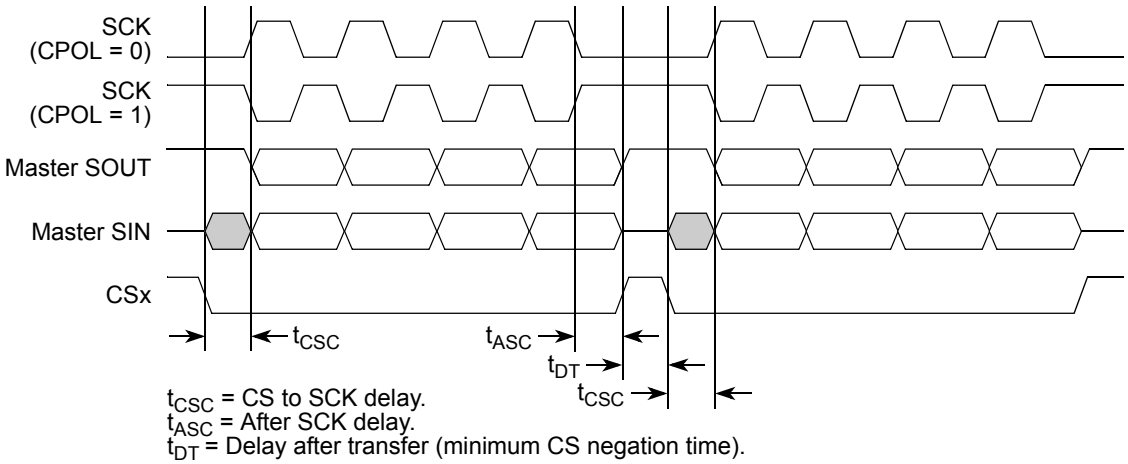


Figure 239. Example of non-continuous format (CPHA = 1, CONT = 0)

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 240 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

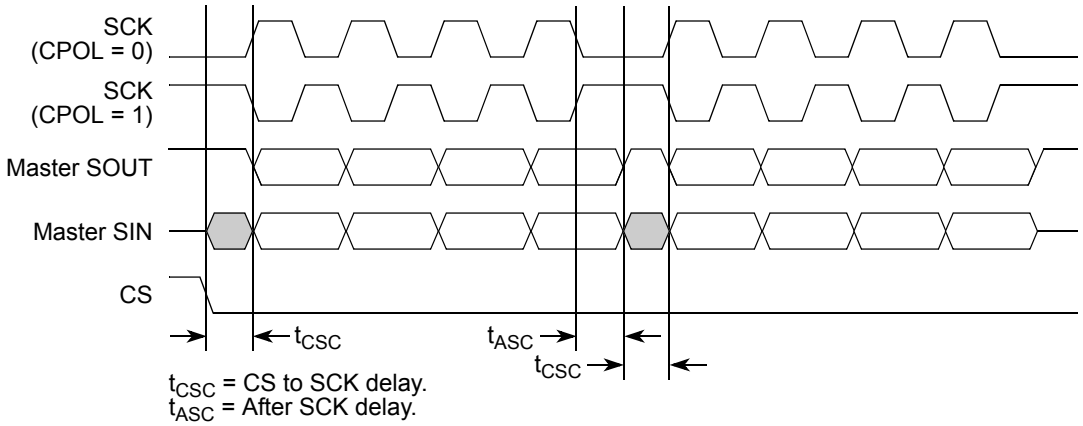


Figure 240. Example of continuous transfer (CPHA = 1, CONT = 1)

In *Figure 240*, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}; that is, 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: You must 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, ensure that the last entry in the TXFIFO, after which TXFIFO becomes empty, has CONT = 0 in the command frame.

When operating in slave mode, ensure that when the last-entry in the TXFIFO is completely transmitted (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave is deselected for any further serial communication; otherwise, 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.

See *Section 23.5.4, DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn)*.

In *Figure 241*, time 'A' shows the one clock interval. Time 'B' is user programmable from a minimum of two system clocks.

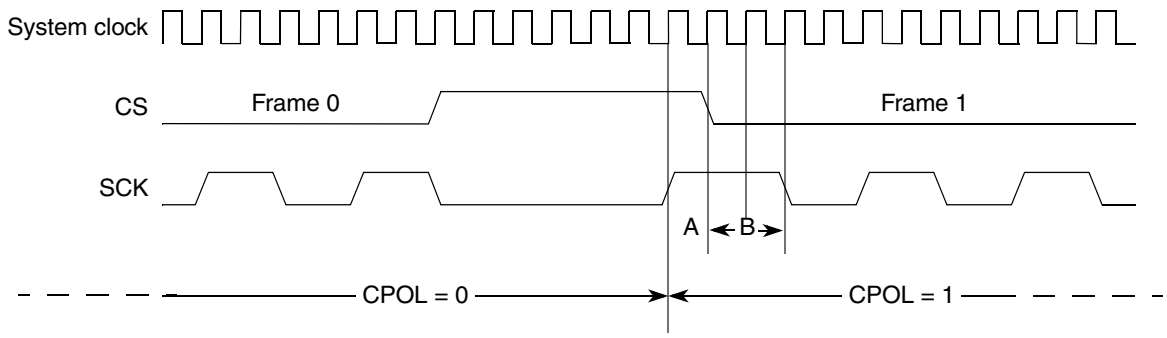


Figure 241. Polarity switching between frames

23.6.6 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:

- The TX FIFO must be cleared before initiating any SPI configuration transfer.
- 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 should be CTAR0.
- 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 242* shows timing diagram for continuous SCK format with continuous selection disabled.

Note: When in Continuous SCK mode, always use CTAR0 for the SPI transfer, and clear the TXFIFO using the MCR[CLR_TXF] field before initiating transfer.

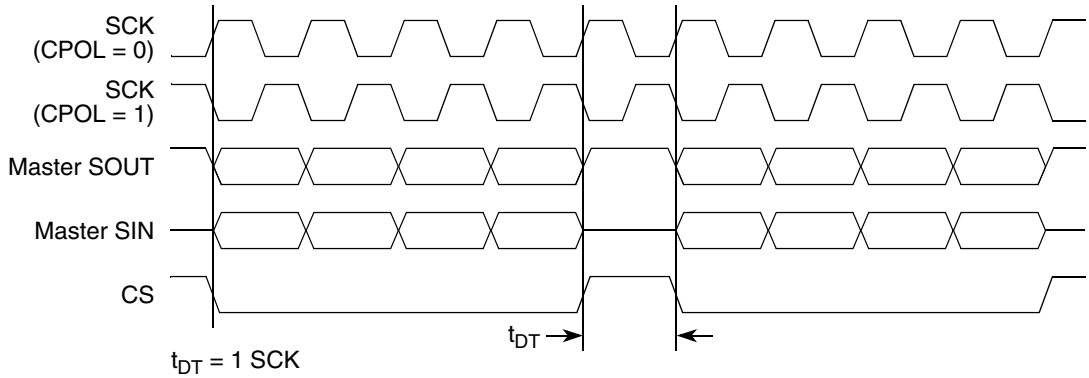


Figure 242. Continuous SCK timing diagram (CONT= 0)

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 243* shows timing diagram for continuous SCK format with continuous selection enabled.

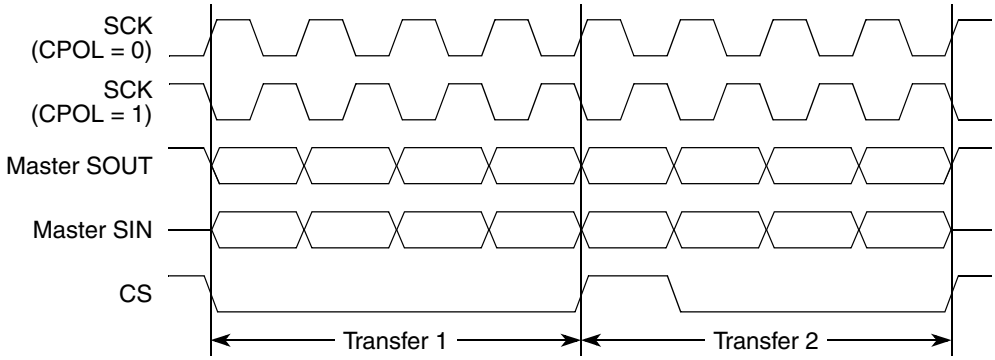


Figure 243. Continuous SCK timing diagram (CONT=1)

23.6.7 Interrupt requests

The DSPI has five conditions that can generate interrupt requests.

Table 253 lists the five conditions.

Table 253. Interrupt request conditions

Condition	Flag
End of transfer queue has been reached (EOQ)	EOQF
Current frame transfer is complete	TCF
TX FIFO underflow has occurred	TFUF
RX FIFO overflow occurred	RFOF
A FIFO overrun occurred(1)	TFUF ORed with RFOF

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 23.5.5, DSPI Status Register (DSPIx_SR)* and the request enable bits are described in the *Section 23.5.6, DSPI Interrupt Request Enable Register (DSPIx_RSER)*. The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt 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. See the EOQ bit description in *Section 23.5.5, DSPI Status Register (DSPIx_SR)*. See *Figure 235* and *Figure 236* that illustrate when EOQF is set.

Transmit FIFO Fill Interrupt 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 is used to generate an interrupt request.

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. See the TCF bit description in *Section 23.5.5, DSPI Status Register (DSPIx_SR)*. See *Figure 235* and *Figure 236* that illustrate 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 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 is used to generate an interrupt request.

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.

23.6.8 Power saving features

The DSPI supports the following 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 request signals cannot be cleared while in the module disable mode.

Slave interface signal gating

The DSPI module enable signal is used to gate slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

23.7 Initialization and application information

23.7.1 How to change 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 change 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. 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.
5. 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.
6. 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.
7. Enable serial transmission and serial reception of data by clearing the EOQF bit.

23.7.2 Baud rate settings

Table 254 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 64 MHz system frequency.

Table 254. Baud rate values

		Baud rate divider prescaler values (DSPI_CTAR[PBR])			
		2	**3**	**5**	**7**
Baud rate scaler values (DSPI_CTAR[BR])	**2**	16.0 MHz	10.7 MHz	6.4 MHz	4.57 MHz
	4	8 MHz	5.33 MHz	3.2 MHz	2.28 MHz
	6	5.33 MHz	3.56 MHz	2.13 MHz	1.52 MHz
	8	4 MHz	2.67 MHz	1.60 MHz	1.15 MHz
	16	2 MHz	1.33 MHz	800 kHz	571 kHz
	32	1 MHz	670 kHz	400 kHz	285 kHz
	64	500 kHz	333 kHz	200 kHz	142 kHz
	128	250 kHz	166 kHz	100 kHz	71.7 kHz
	256	125 kHz	83.2 kHz	50 kHz	35.71 kHz
	512	62.5 kHz	41.6 kHz	25 kHz	17.86 kHz
	1024	31.2 kHz	20.8 kHz	12.5 kHz	8.96 kHz
	2048	15.6 kHz	10.4 kHz	6.25 kHz	4.47 kHz
	4096	7.81 kHz	5.21 kHz	3.12 kHz	2.23 kHz
	8192	3.90 kHz	2.60 kHz	1.56 kHz	1.11 kHz
	16384	1.95 kHz	1.31 kHz	781 Hz	558 Hz
	32768	979 Hz	653 Hz	390 Hz	279 Hz

23.7.3 Delay settings

Table 255 shows the values for the delay after transfer (t_{DT}) that can be generated based on the prescaler values and the scaler values set in the DSPI*x*_CTARs. The values calculated assume a 64 MHz system frequency.

Table 255. Delay values

		Delay prescaler values (DSPI_CTAR[PDT])			
		1	**3**	**5**	**7**
Delay scaler values (DSPI_CTAR[DT])	**2**	31.25 ns	93.75 ns	156.25 ns	218.75 ns
	4	62.5 ns	187.5 ns	312.5 ns	437.5 ns
	8	125 ns	375 ns	625 ns	875 ns
	16	250 ns	750 ns	1.25 µs	1.75 µs
	32	0.5 µs	1.5 µs	2.5 µs	3.5 µs
	64	1 µs	3 µs	5 µs	7 µs
	128	2 µs	6 µs	10 µs	14 µs
	256	4 µs	12 µs	20 µs	28 µs
	512	8 µs	24 µs	40 µs	56 µs
	1024	16 µs	48 µs	80 µs	112 µs
	2048	32 µs	96 µs	160 µs	224 µs
	4096	64 µs	192 µs	320 µs	448 µs
	8192	128 µs	384 µs	640 µs	896 µs
	16384	256 µs	768 µs	1.28 ms	1.79 ms
	32768	512 µs	1.54 ms	2.56 ms	3.58 ms
	65536	1.02 ms	3.07 ms	5.12 ms	7.17 ms

23.7.4 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).

See *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 244 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

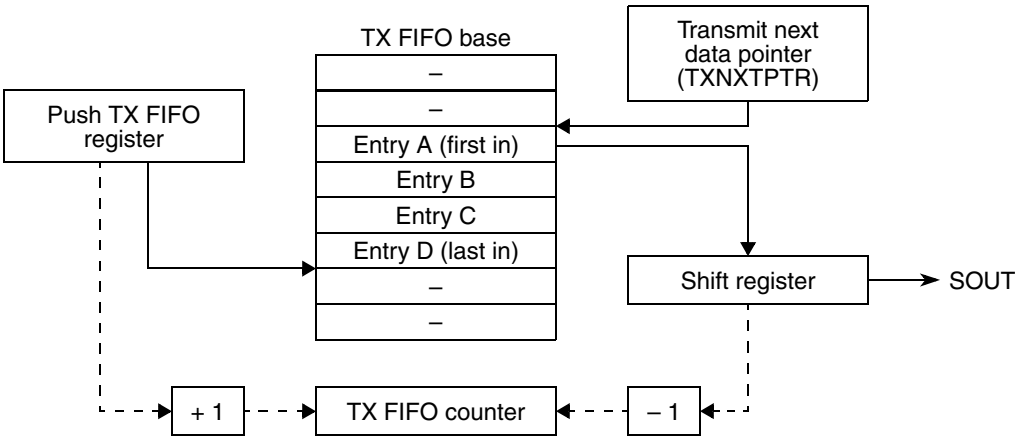


Figure 244. TX FIFO pointers and counter

Address calculation for the first-in entry and last-in entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{TXFIFO base} + 4\ (\text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{TXFIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXTPTR} - 1) \text{ modulo TXFIFO depth}]$$

where:

TXFIFO base = base address of transmit FIFO

TXCTR = transmit FIFO counter

TXNXTPTR = transmit next pointer

TX FIFO depth = transmit FIFO depth, implementation specific

Address calculation for the first-in entry and last-in entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{RXFIFO base} + 4 \times (\text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{RXFIFO base} + 4 \times [(\text{RXCTR} + \text{POPNXTPTR} - 1) \text{ modulo RXFIFO depth}]$$

where:

RXFIFO base = base address of receive FIFO

RXCTR = receive FIFO counter

POPNXTPTR = pop next pointer

RX FIFO depth = receive FIFO depth, implementation specific

24 Timers

24.1 Introduction

This chapter describes the timer modules implemented on the microcontroller:

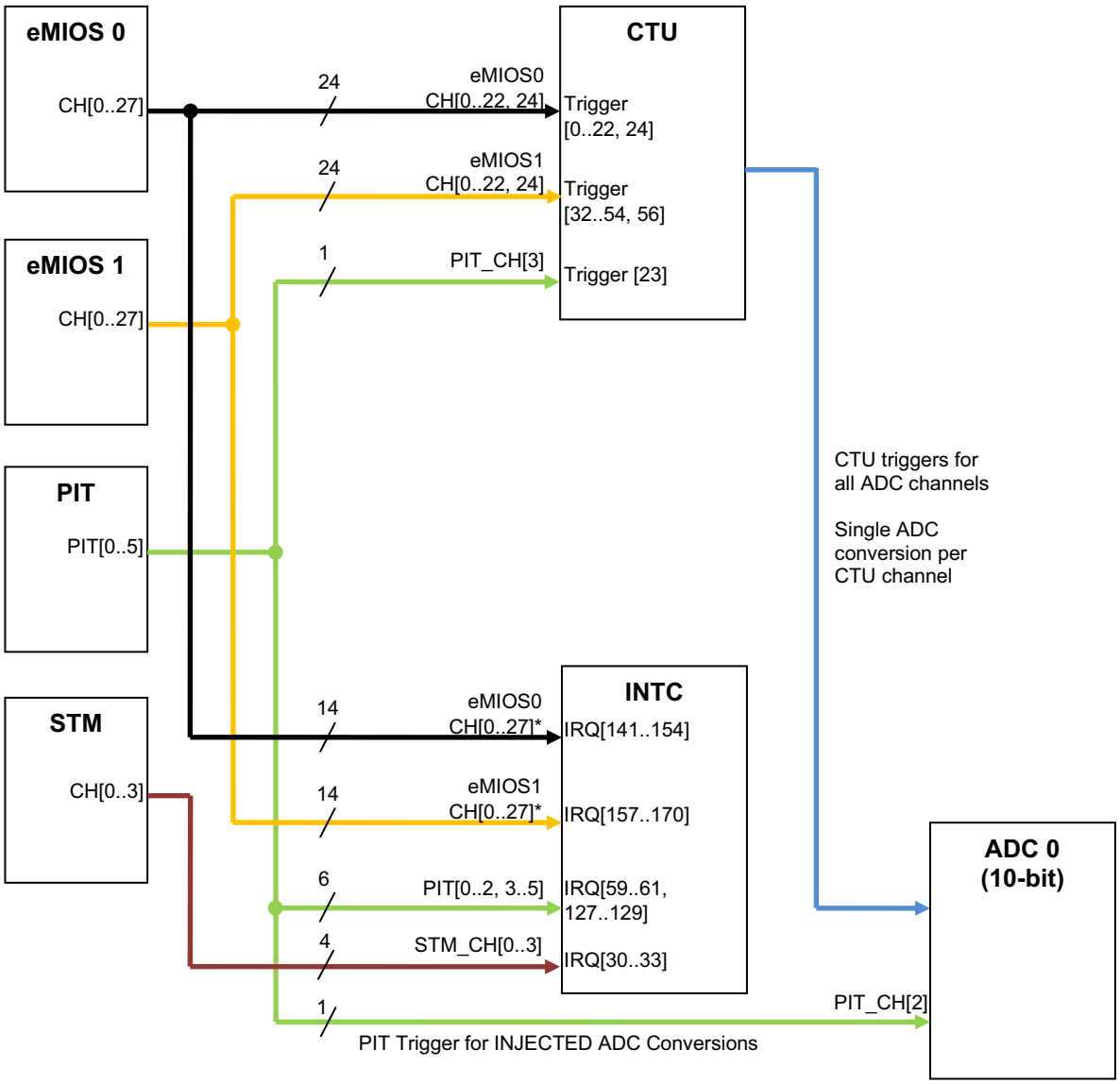
- *System Timer Module (STM)*
- *Enhanced Modular IO Subsystem (eMIOS)*
- *Periodic Interrupt Timer (PIT)*

The microcontroller also has a *Real Time Clock / Autonomous Periodic Interrupt (RTC/API)* module. The main purpose of this is to provide a periodic device wakeup source.

24.2 Technical overview

This section gives a technical overview of each of the timers as well as detailing the pins that can be used to access the timer peripherals if applicable.

Figure 245 details the interaction between the timers and the eDMA, INTC, CTU, and ADC.



Note*

There are 14 interrupt requests from the eMIOS to the INTC. eMIOS channels are routed to the interrupt controller in pairs for example CH[0,1] CH[2,3]

Figure 245. Interaction between timers and relevant peripherals

24.2.1 Overview of the STM

The STM is a 32-bit free running up-counter clocked by the system clock with a configurable 8-bit clock pre-scaler (divide by 1 to 256). The counter is disabled out of reset and must therefore be enabled by software prior to use. The counter value can be read at any time.

The STM has four 32-bit compare channels. Each channel can generate a unique interrupt on an exact match event with the free running counter.

The STM is often used to analyse code execution times. By starting the STM and reading the timer before and after a task or function, you can make an accurate measurement of the time taken in clock cycles to perform the task.

The STM can be configured to stop (freeze) or continue to run in debug mode and is available for use in all operating mode where the system clock is present (not STANDBY or certain STOP mode configurations)

There are no external pins associated with the STM.

24.2.2 Overview of the eMIOS

Each eMIOS offers a combination of PWM, Output Capture and Input Compare functions. There are different types of channel implemented and not every channel supports every eMIOS function. The channel functionality also differs between each eMIOS module. See *Section 24.4, Enhanced Modular IO Subsystem (eMIOS)*, for more details.

Each channel has its own independent 16-bit counter. To allow synchronization between channels, there are a number of shared counter busses that can be used as a common timing reference. These counter buses can be used in combination with the individual channel counters to provide advanced features such as centre aligned PWM with dead time insertion.

Once configured, the eMIOS needs very little CPU intervention. Interrupts, eDMA requests and CTU trigger requests can be raised based on eMIOS flag and timeout events.

The eMIOS is clocked from the system clock via peripheral clock group 3 (with a maximum permitted clock frequency of 64 MHz). The eMIOS can be used in all modes where the system clock is available (which excludes STANDBY mode and STOP mode when the system clock is turned off). The eMIOS has an option to allow the eMIOS counters to freeze or to continue running in debug mode.

The CTU allows an eMIOS event to trigger a single ADC conversion via the CTU without any CPU intervention. Without the CTU, the eMIOS would have to trigger an interrupt request. The respective ISR would then perform a software triggered ADC conversion. This not only uses CPU resource, but also increases the latency between the eMIOS event and the ADC trigger.

The eMIOS "Output Pulse Width Modulation with Trigger" mode (see *Section , Output Pulse Width Modulation with Trigger (OPWMT) mode*) allows a customisable trigger point to be defined at any point in the waveform period. This is extremely useful for LED lighting applications where the trigger can be set to a point where the PWM output is high but after the initial inrush current to the LED has occurred. The PWM trigger can then cause the CTU to perform a single ADC conversion which in turn measures the operating conditions of the LED to ensure it is working within specification. A watchdog feature on the ADC allows channels to be monitored and if the results fall outwith a specific range an interrupt is triggered. This means that all of the measurement is without CPU intervention if the results are within range.

To make it easier to plan which pins to use for the eMIOS, *Table 256* and *Table 257* show the eMIOS channel numbers that are available on each pin. The color shading matches the channel configuration diagram in the eMIOS section.

Table 256. eMIOS_0 channel to pin mapping

Channel	Pin function			Channel	Pin function		
	ALT1	ALT2	ALT3		ALT1	ALT2	ALT3
UC[0]	PA[0]			UC[16]	PE[0]		
UC[1]	PA[1]			UC[17]	PE[1]		
UC[2]	PA[2]			UC[18]	PE[2]		
UC[3]	PA[3], PB[11]			UC[19]	PE[3]		
UC[4]	PA[4], PB[12]			UC[20]	PE[4]		
UC[5]	PA[5], PB[13]			UC[21]	PE[5]		
UC[6]	PA[6], PB[14]			UC[22]	PE[6], PF[5]	PE[8]	
UC[7]	PA[7], PB[15]			UC[23]	PE[7], PF[6]	PE[9]	
UC[8]	PA[8]			UC[24]	PG[10]	PD[12]	
UC[9]	PA[9]			UC[25]	PG[11]	PD[13]	
UC[10]	PA[10], PF[0]			UC[26]	PG[12]	PD[14]	
UC[11]	PA[11], PF[1]			UC[27]	PG[13]	PD[15]	
UC[12]	PC[12], PF[2]						
UC[13]	PC[13], PF[3]						
UC[14]	PC[14], PF[4]						
UC[15]	PC[15]						

Table 257. eMIOS_1 channel to pin mapping

Channel	Pin function			Channel	Pin function		
	ALT1	ALT2	ALT3		ALT1	ALT2	ALT3
UC[0]	PG[14]			UC[16]	PG[7]		
UC[1]	PG[15]			UC[17]	PG[8]		
UC[2]	PH[0]			UC[18]	PG[9]		
UC[3]	PH[1]			UC[19]		PE[12]	
UC[4]	PH[2]			UC[20]		PE[13]	
UC[5]	PH[3]		PH[11]	UC[21]		PE[14]	
UC[6]	PH[4]			UC[22]		PE[15]	
UC[7]	PH[5]			UC[23]		PG[0]	
UC[8]	PH[6]			UC[24]		PG[1]	
UC[9]	PH[7]			UC[25]	PF[12]		
UC[10]	PH[8]			UC[26]	PF[13]		
UC[11]	PG[2]			UC[27]		PF[14]	
UC[12]	PG[3]						
UC[13]	PG[4]						
UC[14]	PG[5]						
UC[15]	PG[6]						

24.2.3 Overview of the PIT

The PIT module consists of 6 Periodic Interrupt Timers (PITs) clocked from the system clock.

Out of reset, the PITis disabled. There is a global disable control bit for all of the PIT timers. Before using the timers, software must clear the appropriate disabled bit. Each of the PIT timers are effectively standalone entities and each have their own timer and control registers.

The PIT timers are 32-bit count down timers. To use them, you must first program an initial value into the LDVAL register. The timer will then start to count down and can be read at any time. Once the timer reaches 0x0000_0000, a flag is set and the previous value is automatically re-loaded into the LDVAL register and the countdown starts again. The flag event can be routed to a dedicated INTC interrupt if desired.

The PIT is also used to trigger other events:

- *1 PIT channels can be used to trigger a CTU ADC conversion (single)*
- *1 PIT channel can be used to directly trigger injected conversions on the ADC*

The timers can be configured to stop (freeze) or to continue to run in debug mode. The PITis available in all modes where a system clock is generated.

There are no external pins associated with the PIT.

24.3 System Timer Module (STM)

24.3.1 Introduction

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).

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*

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 register. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

24.3.2 External signal description

The STM does not have any external interface signals.

24.3.3 Memory map and register definition

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.

Memory map

The STM memory map is shown in *Table 258*.

Table 258. STM memory map

Base address: 0xFFF3_C000		
Address offset	**Register**	**Location**
0x0000	STM Control Register (STM_CR)	*on page 24-519*
0x0004	STM Counter Value (STM_CNT)	*on page 24-519*
0x0008–0x000C	Reserved	
0x0010	STM Channel 0 Control Register (STM_CCR0)	*on page 24-520*
0x0014	STM Channel 0 Interrupt Register (STM_CIR0)	*on page 24-520*
0x0018	STM Channel 0 Compare Register (STM_CMP0)	*on page 24-521*
0x001C	Reserved	
0x0020	STM Channel 1 Control Register (STM_CCR1)	*on page 24-520*
0x0024	STM Channel 1 Interrupt Register (STM_CIR1)	*on page 24-520*
0x0028	STM Channel 1 Compare Register (STM_CMP1)	*on page 24-521*
0x002C	Reserved	
0x0030	STM Channel 2 Control Register (STM_CCR2)	*on page 24-520*
0x0034	STM Channel 2 Interrupt Register (STM_CIR2)	*on page 24-520*
0x0038	STM Channel 2 Compare Register (STM_CMP2)	*on page 24-521*
0x003C	Reserved	
0x0040	STM Channel 3 Control Register (STM_CCR3)	*on page 24-520*
0x0044	STM Channel 3 Interrupt Register (STM_CIR3)	*on page 24-520*
0x0048	STM Channel 3 Compare Register (STM_CMP3)	*on page 24-521*
0x004C–0x3FFF	Reserved	

Register descriptions

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 246. STM Control Register (STM_CR)

Offset: 0x000 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	CPS								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 259. STM_CR field descriptions

Field	Description
CPS	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 247. STM Count Register (STM_CNT)

Offset: 0x004 Access: Read/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	CNT							
W								
Reset	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

Table 260. 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_CCRn)

The STM Channel Control Register (STM_CCRn) has the enable bit for channel n of the timer.

Figure 248. STM Channel Control Register (STM_CCRn)

Offset: 0x10+0x10*n 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	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 261. STM_CCRn field descriptions

Field	Description
CEN	Channel Enable. 0 = The channel is disabled. 1 = The channel is enabled.

STM Channel Interrupt Register (STM_CIRn)

The STM Channel Interrupt Register (STM_CIRn) has the interrupt flag for channel n of the timer.

Figure 249. STM Channel Interrupt Register (STM_CIRn)

Offset: 0x14+0x10*n 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	CIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 262. STM_CIRn field descriptions

Field	Description
CIF	Channel Interrupt Flag 0 = No interrupt request. 1 = Interrupt request due to a match on the channel.

STM Channel Compare Register (STM_CMPn)

The STM channel compare register (STM_CMPn) holds the compare value for channel n.

Figure 250. STM Channel Compare Register (STM_CMPn)

Offset: 0x18+0x10*n Access: Read/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	CMP							
W								
Reset	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

Table 263. STM_CMPn field descriptions

Field	Description
CMP	Compare value for channel n. If the STM_CCRn[CEN] bit is set and the STM_CMPn register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIRn[CIF] bit is set.

24.3.4 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_CCRn), a channel interrupt register (STM_CIRn) and a channel compare register (STM_CMPn). The channel is enabled by setting the STM_CCRn[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_CIRn[CIF] bit. A write of 0 to the STM_CIRn[CIF] bit has no effect.

Note: STM counter does not advance when the system clock is stopped.

24.4 Enhanced Modular IO Subsystem (eMIOS)

24.4.1 Introduction

Overview of the eMIOS module

The eMIOS provides functionality to generate or measure time events. Each channel provides a subset of the functionality available in the unified channel, at a resolution of 16 bits, and provides a user interface that is consistent with previous eMIOS implementations.

Features of the eMIOS module

- *2 eMIOS blocks with 28 channels each*
  - 50 channels with OPWMT, which can be connected to the CTU
  - 6 channels with single action IC/OC
  - Both eMIOS blocks can be synchronized
- *1 global prescaler*
- *16-bit data registers*
- *10 x 16-bit wide counter buses*
  - Counter buses B, C, D, and E can be driven by Unified Channel 0, 8, 16, and 24, respectively
  - Counter bus A is driven by the Unified Channel #23
  - Several channels have their own time base, alternative to the counter buses
  - Shared timebases through the counter buses
  - Synchronization among timebases
- *Control and Status bits grouped in a single register*
- *Shadow FLAG register*
- *State of the UC can be frozen for debug purposes*
- *Motor control capability*

Modes of operation

The Unified Channels can be configured to operate in the following modes:

- *General purpose input/output*
- *Single Action Input Capture*
- *Single Action Output Compare*
- *Input Pulse Width Measurement*
- *Input Period Measurement*
- *Double Action Output Compare*
- *Modulus Counter*
- *Modulus Counter Buffered*
- *Output Pulse Width and Frequency Modulation Buffered*
- *Output Pulse Width Modulation Buffered*
- *Output Pulse Width Modulation with Trigger*
- *Center Aligned Output Pulse Width Modulation Buffered*

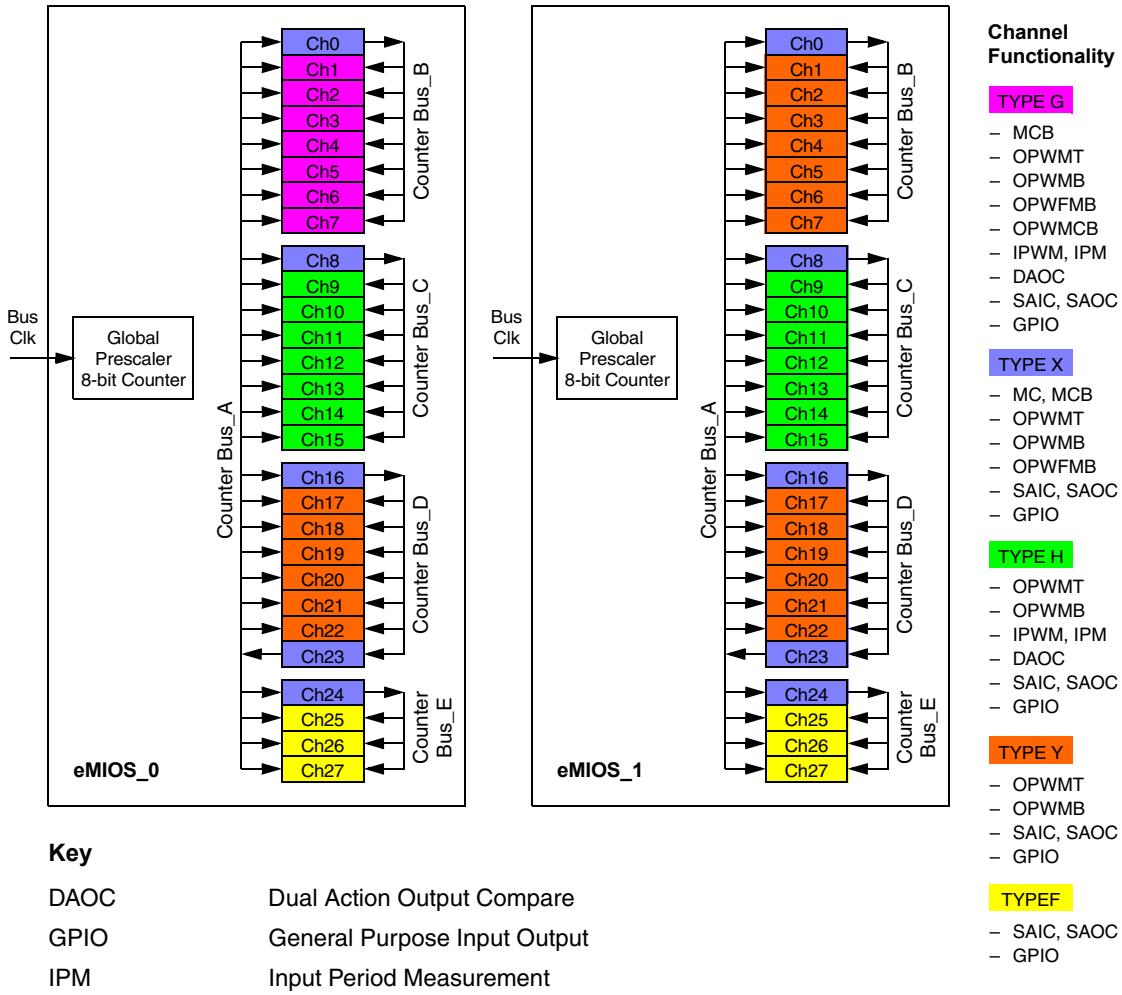
These modes are described in *Section , UC modes of operation*.

Each channel can have a specific set of modes implemented, according to device requirements.

If an unimplemented mode (reserved) is selected, the results are unpredictable such as writing a reserved value to MODE[0:6] in *Section , eMIOS UC Control Register (EMIOSC[n])*.

Channel implementation

Figure 251 shows the channel configuration of the eMIOS blocks.



Key

DAOC	Dual Action Output Compare
GPIO	General Purpose Input Output
IPM	Input Period Measurement
IPWM	Input Pulse Width Measurement
MC	Modulus Counter
MCB	Buffered Modulus Counter
OPWMB	Buffered Output Pulse Width Modulation
OPWMT	Buffered Output Pulse Width Modulation with Trigger
OPWFMB	Buffered Output Pulse Width and Frequency Modulation
OPWMCB	Center Aligned Output PWM Buffered with Dead-Time
SAIC	Single Action Input Capture
SAOC	Single Action Output Compare

Figure 251. Channel configuration

Channel mode selection

Channel modes are selected using the mode selection bits MODE[0:6] in the eMIOS UC Control Register (EMIOSC[n]). *Table 276* provides the specific mode selection settings for the eMIOS implementation on this device.

24.4.2 External signal description

For information on eMIOS external signals on this device, please refer to the signal description chapter of the reference manual.

24.4.3 Memory map and register description

Memory maps

The overall address map organization is shown in *Table 264*.

Unified Channel memory map

Table 264. eMIOS memory map

Base addresses: 0xC3FA_0000 (eMIOS_0) 0xC3FA_4000 (eMIOS_1)		
Address offset	**Description**	**Location**
0x000–0x003	eMIOS Module Configuration Register (EMIOSMCR)	*on page 24-526*
0x004–0x007	eMIOS Global FLAG (EMIOSGFLAG) Register	*on page 24-527*
0x008–0x00B	eMIOS Output Update Disable (EMIOSOUDIS) Register	*on page 24-528*
0x00C–0x00F	eMIOS Disable Channel (EMIOSUCDIS) Register	*on page 24-529*
0x010–0x01F	Reserved	—
0x020–0x11F	Channel [0] to Channel [7]	—
0x120–0x21F	Channel [8] to Channel [15]	—
0x220–0x31F	Channel [16] to Channel [23]	—
0x320–0x39F	Channel [24] to Channel [27]	—
0x3A0–0xFFF	Reserved	—

Addresses of Unified Channel registers are specified as offsets from the channel's base address; otherwise the eMIOS base address is used as reference.

Table 265 describes the Unified Channel memory map.

Table 265. Unified Channel memory map

UC[n] base address	Description	Location
0x00	eMIOS UC A Register (EMIOSA[n])	*on page 24-529*
0x04	eMIOS UC B Register (EMIOSB[n])	*on page 24-530*
0x08	eMIOS UC Counter Register (EMIOSCNT[n])	*on page 24-531*
0x0C	eMIOS UC Control Register (EMIOSC[n])	*on page 24-531*
0x10	eMIOS UC Status Register (EMIOSS[n])	*on page 24-536*
0x14	eMIOS UC Alternate A Register (EMIOSALTA[n])	*on page 24-537*
0x18–0x1F	Reserved	—

Register description

All control registers are 32 bits wide. Data registers and counter registers are 16 bits wide.

eMIOS Module Configuration Register (EMIOSMCR)

The EMIOSMCR contains global control bits for the eMIOS block.

Figure 252. eMIOS Module Configuration Register (EMIOSMCR)

Address: eMIOS base address +0x00

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MDIS	FRZ	GTBE	0	GPREN	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	GPRE								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 266. EMIOSMCR field descriptions

Field	Description
MDIS	Module Disable Puts the eMIOS in low power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOSMCR, EMIOSOUDIS and EMIOSUCDIS. 1 = Enter low power mode 0 = Clock is running

Table 266. EMIOSMCR field descriptions (continued)

Field	Description
FRZ	Freeze Enables the eMIOS to freeze the registers of the Unified Channels when Debug Mode is requested at MCU level. Each Unified Channel should have FREN bit set in order to enter freeze state. While in Freeze state, the eMIOS continues to operate to allow the MCU access to the Unified Channels registers. The Unified Channel will remain frozen until the FRZ bit is written to '0' or the MCU exits Debug mode or the Unified Channel FREN bit is cleared. 1 = Stops Unified Channels operation when in Debug mode and the FREN bit is set in the EMIOSC[n] register 0 = Exit freeze state
GTBE	Global Time Base Enable The GTBE bit is used to export a Global Time Base Enable from the module and provide a method to start time bases of several blocks simultaneously. 1 = Global Time Base Enable Out signal asserted 0 = Global Time Base Enable Out signal negated *Note: The Global Time Base Enable input pin controls the internal counters. When asserted, Internal counters are enabled. When negated, Internal counters disabled.*
GPREN	Global Prescaler Enable The GPREN bit enables the prescaler counter. 1 = Prescaler enabled 0 = Prescaler disabled (no clock) and prescaler counter is cleared
GPRE	Global Prescaler The GPRE bits select the clock divider value for the global prescaler, as shown in *Table 267*.

Table 267. Global prescaler clock divider

GPRE	Divide ratio
00000000	1
00000001	2
00000010	3
00000011	4
. . . .	. . . .
11111110	255
11111111	256

eMIOS Global FLAG (EMIOSGFLAG) Register

The EMIOSGFLAG is a read-only register that groups the flag bits (F[27:0]) from all channels. This organization improves interrupt handling on simpler devices. Each bit relates to one channel.

For Unified Channels these bits are mirrors of the FLAG bits in the EMIOSS[n] register.

Figure 253. eMIOS Global FLAG (EMIOSGFLAG) Register

Address: eMIOS base address +0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
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	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 268. EMIOSGFLAG field descriptions

Field	Description
Fn	Channel [n] Flag bit

eMIOS Output Update Disable (EMIOSOUDIS) Register

Figure 254. eMIOS Output Update Disable (EMIOSOUDIS) Register

Address: eMIOS base address +0x08

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	OU27	OU26	OU25	OU24	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
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	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 269. EMIOSOUDIS field descriptions

Field	Description
OUn	Channel [n] Output Update Disable bit When running MC, MCB or an output mode, values are written to registers A2 and B2. OU[n] bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 1 = Transfers disabled 0 = Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately.

eMIOS Disable Channel (EMIOSUCDIS) Register

Figure 255. eMIOS Enable Channel (EMIOSUCDIS) Register

Address: eMIOS base address +0x0C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	CHDIS27	CHDIS26	CHDIS25	CHDIS24	CHDIS23	CHDIS22	CHDIS21	CHDIS20	CHDIS19	CHDIS18	CHDIS17	CHDIS16
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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHDIS15	CHDIS14	CHDIS13	CHDIS12	CHDIS11	CHDIS10	CHDIS9	CHDIS8	CHDIS7	CHDIS6	CHDIS5	CHDIS4	CHDIS3	CHDIS2	CHDIS1	CHDIS0
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

Table 270. EMIOSUCDIS field descriptions

Field	Description
CHDISn	Enable Channel [n] bit The CHDIS[n] bit is used to disable each of the channels by stopping its respective clock. 1 = Channel [n] disabled 0 = Channel [n] enabled

eMIOS UC A Register (EMIOSA[n])

Figure 256. eMIOS UC A Register (EMIOSA[n])

Address: UC[n] base address + 0x00

	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	A															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

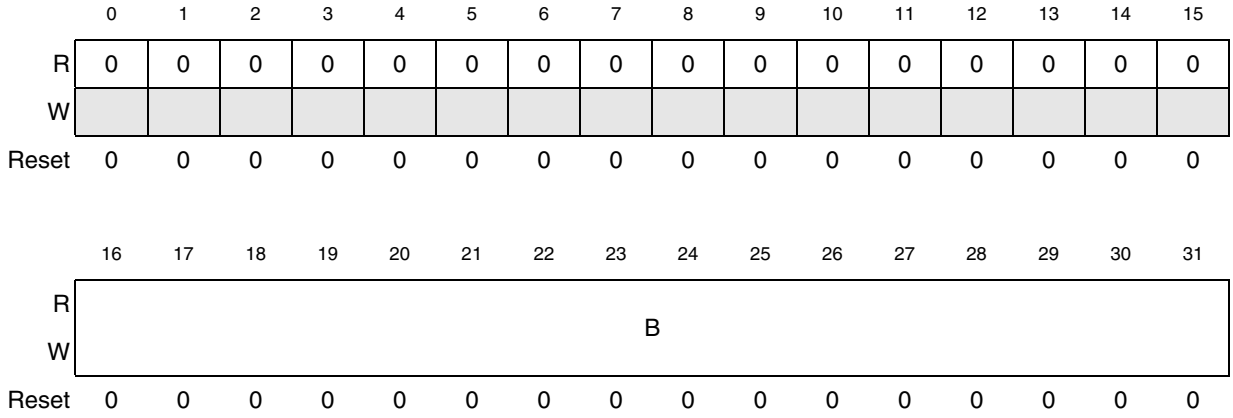
Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOSA[n]. Both A1 and A2 are cleared by reset.

Figure 271 summarizes the EMIOSA[n] writing and reading accesses for all operation modes. For more information see *Section , UC modes of operation*.

eMIOS UC B Register (EMIOSB[n])

Figure 257. eMIOS UC B Register (EMIOSB[n])

Address: UC[n] base address + 0x04



Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOSB[n]. Both B1 and B2 are cleared by reset. *Table 271* summarizes the EMIOSB[n] writing and reading accesses for all operation modes. For more information see *Section , UC modes of operation*.

Depending on the channel configuration, it may have EMIOSB register or not. This means that, if at least one mode that requires the register is implemented, then the register is present; otherwise it is absent.

Table 271. EMIOSA[n], EMIOSB[n] and EMIOSALTA[n] values assignment

Operation mode	Register access					
	write	read	write	read	alt write	alt read
GPIO	A1, A2	A1	B1,B2	B1	A2	A2
SAIC(1)	—	A2	B2	B2	—	—
SAOC(1)	A2	A1	B2	B2	—	—
IPWM	—	A2	—	B1	—	—
IPM	—	A2	—	B1	—	—
DAOC	A2	A1	B2	B1	—	—
MC(1)	A2	A1	B2	B2	—	—
OPWMT	A1	A1	B2	B1	A2	A2
MCB(1)	A2	A1	B2	B2	—	—
OPWFMB	A2	A1	B2	B1	—	—
OPWMCB	A2	A1	B2	B1	—	—
OPWMB	A2	A1	B2	B1	—	—

1. In these modes, the register EMIOSB[n] is not used, but B2 can be accessed.

eMIOS UC Counter Register (EMIOSCNT[n])

Figure 258. eMIOS UC Counter Register (EMIOSCNT[n])

Address: UC[n] base address + 0x08

	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(1)																
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	C															
W(1)																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. In GPIO mode or Freeze action, this register is writable.

The EMIOSCNT[n] register contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the EMIOSCNT[n] register is read/write. For all others modes, the EMIOSCNT[n] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to *Section , UC modes of operation* for details).

Depending on the channel configuration it may have an internal counter or not. It means that if at least one mode that requires the counter is implemented, then the counter is present; otherwise it is absent.

Channels of type X and G have the internal counter enabled, so their timebase can be selected by channel's BSL[1:0]=11:eMIOS_A - channels 0 to 8, 16, 23 and 24, eMIOS_B = channels 0, 8, 16, 23 and 24. Other channels from the above list don't have internal counters.

eMIOS UC Control Register (EMIOSC[n])

The Control register gathers bits reflecting the status of the UC input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Figure 259. eMIOS UC Control Register (EMIOSC[n])

Address: UC[n] base address + 0x0C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	0	0	0	UCPRE		UCPREN	DMA	0	IF				FCK	FEN	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	BSL		EDSEL	EDPOL	MODE						
W			FORCMA	FORCMB												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 272. EMIOSC[n] field descriptions

Field	Description
FREN	Freeze Enable bit The FREN bit, if set and validated by FRZ bit in EMIOSMCR register allows the channel to enter freeze state, freezing all registers values when in debug mode and allowing the MCU to perform debug functions. 1 = Freeze UC registers values 0 = Normal operation
UCPRE	Prescaler bits The UCPRE bits select the clock divider value for the internal prescaler of Unified Channel, as shown in *Table 273*.
UCPREN	Prescaler Enable bit The UCPREN bit enables the prescaler counter. 1 = Prescaler enabled 0 = Prescaler disabled (no clock)
DMA	Direct Memory Access bit The DMA bit selects if the FLAG generation will be used as an interrupt or as a CTU trigger. 1 = Flag/overrun assigned to CTU trigger 0 = Flag/overrun assigned to interrupt request
IF	Input Filter The IF field controls the programmable input filter, selecting the minimum input pulse width that can pass through the filter, as shown in *Table 274*. For output modes, these bits have no meaning.
FCK	Filter Clock select bit The FCK bit selects the clock source for the programmable input filter. 1 = Main clock 0 = Prescaled clock

Table 272. EMIOSC[n] field descriptions (continued)

Field	Description
FEN	FLAG Enable bit The FEN bit allows the Unified Channel FLAG bit to generate an interrupt signal or a CTU trigger signal (The type of signal to be generated is defined by the DMA bit). 1 = Enable (FLAG will generate an interrupt request or a CTU trigger) 0 = Disable (FLAG does not generate an interrupt request or a CTU trigger)
FORCMA	Force Match A bit For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect. 1 = Force a match at comparator A 0 = Has no effect *Note: For input modes, the FORCMA bit is not used and writing to it has no effect.*
FORCMB	Force Match B bit For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect. 1 = Force a match at comparator B 0 = Has not effect *Note: For input modes, the FORCMB bit is not used and writing to it has no effect.*
BSL	Bus Select The BSL field is used to select either one of the counter buses or the internal counter to be used by the Unified Channel. Refer to *Table 275* for details.
EDSEL	Edge Selection bit For input modes, the EDSEL bit selects whether the internal counter is triggered by both edges of a pulse or just by a single edge as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect. 1 = Both edges triggering 0 = Single edge triggering defined by the EDPOL bit For GPIO in mode, the EDSEL bit selects if a FLAG can be generated. 1 = No FLAG is generated 0 = A FLAG is generated as defined by the EDPOL bit For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match. 1 = The output flip-flop is toggled 0 = The EDPOL value is transferred to the output flip-flop
EDPOL	Edge Polarity bit For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect. 1 = Trigger on a rising edge 0 = Trigger on a falling edge For output modes, the EDPOL bit is used to select the logic level on the output pin. 1 = A match on comparator A sets the output flip-flop, while a match on comparator B clears it 0 = A match on comparator A clears the output flip-flop, while a match on comparator B sets it

Table 272. EMIOSC[n] field descriptions (continued)

Field	Description
MODE	Mode selection The MODE field selects the mode of operation of the Unified Channel, as shown in *Table 276*. *Note: If a reserved value is written to mode the results are unpredictable.*

Table 273. UC internalprescaler clock divider

UCPRE	Divide ratio
00	1
01	2
10	3
11	4

Table 274. UC input filter bits

IF(1)	Minimum input pulse width [FLT_CLK periods]
0000	Bypassed(2)
0001	02
0010	04
0100	08
1000	16
all others	Reserved

1. Filter latency is 3 clock edges.
2. The input signal is synchronized before arriving to the digital filter.

Table 275. UC BSL bits

BSL	Selected bus
00	All channels: counter bus[A]
01	Channels 0 to 7: counter bus[B] Channels 8 to 15: counter bus[C] Channels 16 to 23: counter bus[D] Channels 24 to 27: counter bus[E]
10	Reserved
11	All channels: internal counter

Table 276. Channel mode selection

MODE(1)	Mode of operation
0000000	General purpose Input/Output mode (input)
0000001	General purpose Input/Output mode (output)
0000010	Single Action Input Capture
0000011	Single Action Output Compare
0000100	Input Pulse Width Measurement
0000101	Input Period Measurement
0000110	Double Action Output Compare (with FLAG set on B match)
0000111	Double Action Output Compare (with FLAG set on both match)
0001000 – 0001111	Reserved
001000b	Modulus Counter (Up counter with clear on match start)
001001b	Modulus Counter (Up counter with clear on match end)
00101bb	Modulus Counter (Up/Down counter)
0011000 – 0100101	Reserved
0100110	Output Pulse Width Modulation with Trigger
0100111 – 1001111	Reserved
101000b	Modulus Counter Buffered (Up counter)
101001b	Reserved
10101bb	Modulus Counter Buffered (Up/Down counter)
10110b0	Output Pulse Width and Frequency Modulation Buffered
10110b1	Reserved
10111b0	Center Aligned Output Pulse Width Modulation Buffered (with trail edge dead-time)
10111b1	Center Aligned Output Pulse Width Modulation Buffered (with lead edge dead-time)
11000b0	Output Pulse Width Modulation Buffered
1100001 – 1111111	Reserved

1. b = adjust parameters for the mode of operation. Refer to *Section , UC modes of operation* for details.

eMIOS UC Status Register (EMIOSS[n])

Figure 260. eMIOS UC Status Register (EMIOSS[n])

Address: UC[n] base address + 0x10

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 277. EMIOSS[n] field descriptions

Field	Description
OVR	Overrun bit The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. 1 = Overrun has occurred 0 = Overrun has not occurred
OVFL	Overflow bit The OVFL bit indicates that an overflow has occurred in the internal counter. OVFL must be cleared by software writing a 1 to the OVFLC bit. 1 = An overflow had occurred 0 = No overflow
UCIN	Unified Channel Input pin bit The UCIN bit reflects the input pin state after being filtered and synchronized.
UCOUT	UCOUT — Unified Channel Output pin bit The UCOUT bit reflects the output pin state.
FLAG	FLAG bit The FLAG bit is set when an input capture or a match event in the comparators occurred. 1 = FLAG set event has occurred 0 = FLAG cleared *Note: When DMA bit is set, the FLAG bit can be cleared by the CTU.*

eMIOS UC Alternate A Register (EMIOSALTA[n])

Figure 261. eMIOS UC Alternate A register (EMIOSALTA[n])

Address: UC[n] base address + 0x14

	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	ALTA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The EMIOSALTA[n] register provides an alternate address to access A2 channel registers in restricted modes (GPIO, OPWMT) only. If EMIOSA[n] register is used along with EMIOSALTA[n], both A1 and A2 registers can be accessed in these modes. *Figure 271* summarizes the EMIOSALTA[n] writing and reading accesses for all operation modes. Please, see *Section , General purpose Input/Output (GPIO) mode*, *Section , Output Pulse Width Modulation with Trigger (OPWMT) mode* for a more detailed description of the use of EMIOSALTA[n] register.

24.4.4 Functional description

The five types of channels of the eMIOS can operate in the modes as listed in *Figure 251*. The eMIOS provides independently operating unified channels (UC) that can be configured and accessed by a host MCU. Up to four time bases can be shared by the channels through four counter buses and each unified channel can generate its own time base. The eMIOS block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

Unified Channel (UC)

Each Unified Channel consists of:

- *Counter bus selector, which selects the time base to be used by the channel for all timing functions*
- *A programmable clock prescaler*
- *Two double buffered data registers A and B that allow up to two input capture and/or output compare events to occur before software intervention is needed.*
- *Two comparators (equal only) A and B, which compares the selected counter bus with the value in the data registers*
- *Internal counter, which can be used as a local time base or to count input events*
- *Programmable input filter, which ensures that only valid pin transitions are received by channel*
- *Programmable input edge detector, which detects the rising, falling or either edges*
- *An output flip-flop, which holds the logic level to be applied to the output pin*
- *eMIOS Status and Control register*

UC modes of operation

The mode of operation of the Unified Channel is determined by the mode select bits MODE[0:6] in the eMIOS UC Control Register (EMIOSC[n]) (see *Figure 259* for details).

As the internal counter EMIOSCNT[n] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

In order to provide smooth waveform generation even if A and B registers are changed on the fly, it is available the MCB, OPWFMB, OPWMB and OPWMCB modes. In these modes A and B registers are double buffered.

General purpose Input/Output (GPIO) mode

In GPIO mode, all input capture and output compare functions of the UC are disabled, the internal counter (EMIOSCNT[n] register) is cleared and disabled. All control bits remain accessible. In order to prepare the UC for a new operation mode, writing to registers EMIOSA[n] or EMIOSB[n] stores the same value in registers A1/A2 or B1/B2, respectively. Writing to register EMIOSALTA[n] stores a value only in register A2.

MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

It is required that when changing MODE[0:6], the application software goes to GPIO mode first in order to reset the UC's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode (MODE[0:6] = 0000000), the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

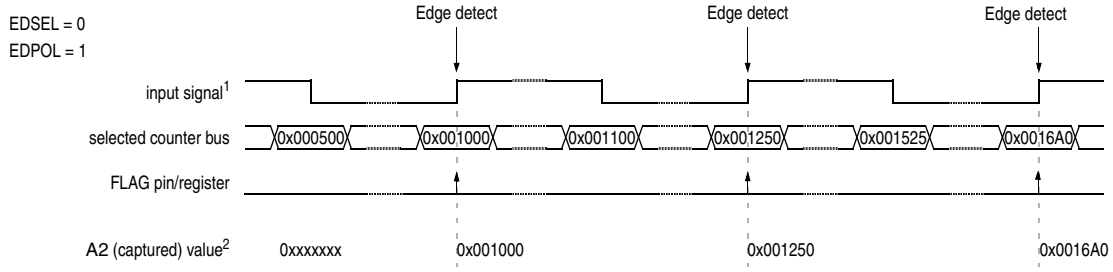
In GPIO output mode (MODE[0:6] = 0000001), the Unified Channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

Single Action Input Capture (SAIC) mode

In SAIC mode (MODE[0:6] = 0000010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. The FLAG bit is set along with the capture event to indicate that an input capture has occurred. Register EMIOSA[n] returns the value of register A2. As soon as the SAIC mode is entered coming out from GPIO mode the channel is ready to capture events. The events are captured as soon as they occur thus reading register A always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from EMIOSA[n] register. The FLAG is set at any time a new event is captured.

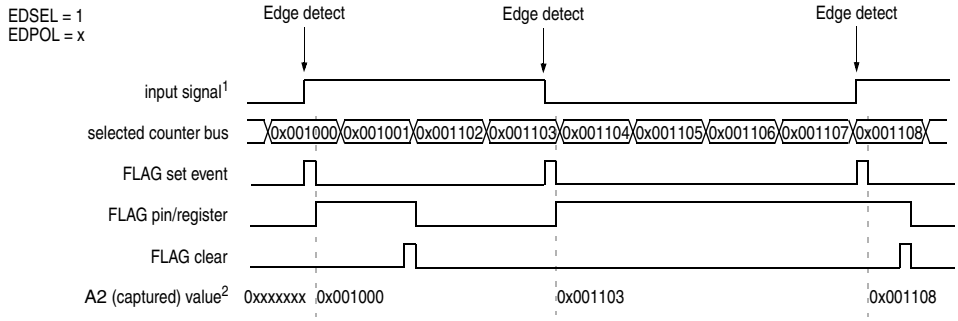
The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOSC[n] register.

Figure 262 and *Figure 263* show how the Unified Channel can be used for input capture.



Notes: 1. After input filter
2. EMIOSA[n] <= A2

Figure 262. Single action input capture with rising edge triggering example



Notes: 1. After input filter
2. EMIOSA[n] <= A2

Figure 263. Single action input capture with both edges triggering example

Single Action Output Compare (SAOC) mode

In SAOC mode (MODE[0:6] = 0000011) a match value is loaded in register A2 and then immediately transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects whether the output flip-flop is toggled or the value in EDPOL is transferred to it. Along with the match the FLAG bit is set to indicate that the output compare match has occurred. Writing to register EMIOSA[n] stores the value in register A2 and reading to register EMIOSA[n] returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in EMIOSC[n] register. In this case, the FLAG bit is not set.

When SAOC mode is entered coming out from GPIO mode the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

Counter bus can be either internal or external and is selected through bits BSL[0:1].

Figure 264 and *Figure 265* show how the Unified Channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode the matches are enabled thus the desired match value on register A1 must be written before the mode is entered. A1 register can be updated at any time thus modifying the match value which will reflect in the output signal generated by the channel. Subsequent matches are enabled with no need of further writes to EMIOSA[n] register. The FLAG is set at the same time a match occurs (see *Figure 266*).

Note: The channel internal counter in SAOC mode is free-running. It starts counting as soon as the SAOC mode is entered.

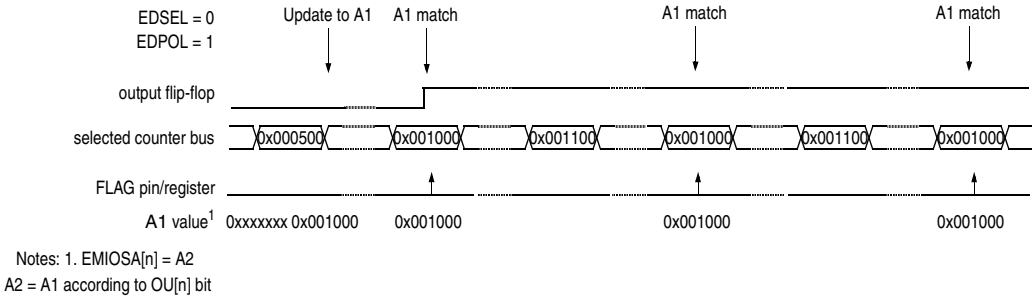


Figure 264. SAOC example with EDPOL value being transferred to the output flip-flop

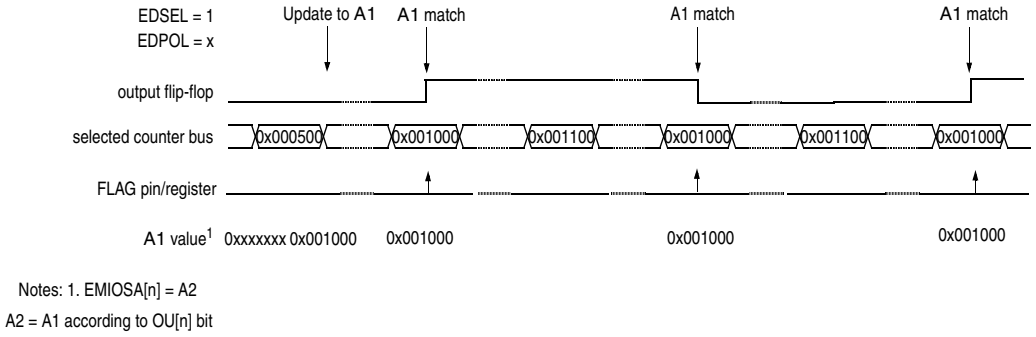
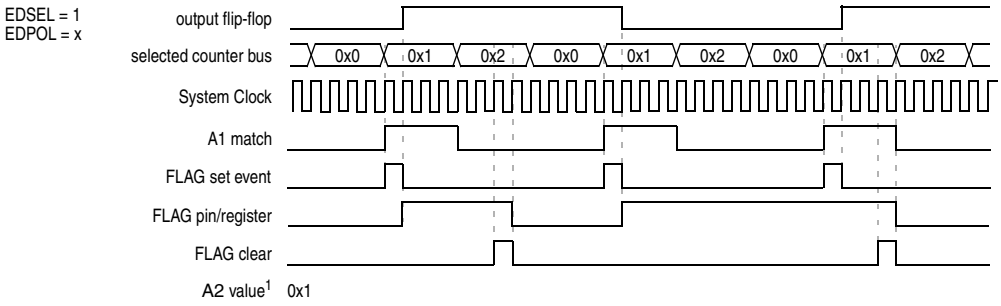


Figure 265. SAOC example toggling the output flip-flop



Note: 1. EMIOSA[n] <= A2

Figure 266. SAOC example with flag behavior

Input Pulse Width Measurement (IPWM) Mode

The IPWM mode (MODE[0:6] = 0000100) allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (that is, pulse polarity) is selected by EDPOL bit in the EMIOSC[n] register. Registers EMIOSA[n] and EMIOSB[n] return the values in register A2 and B1, respectively.

The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1 and A1 will be updated with the latest captured values and the FLAG will remain set. Registers EMIOSA[n] and EMIOSB[n] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOSA[n] forces B1 be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of EMIOSB[n] register. Reading EMIOSB[n] register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 267 shows how the Unified Channel can be used for input pulse width measurement.

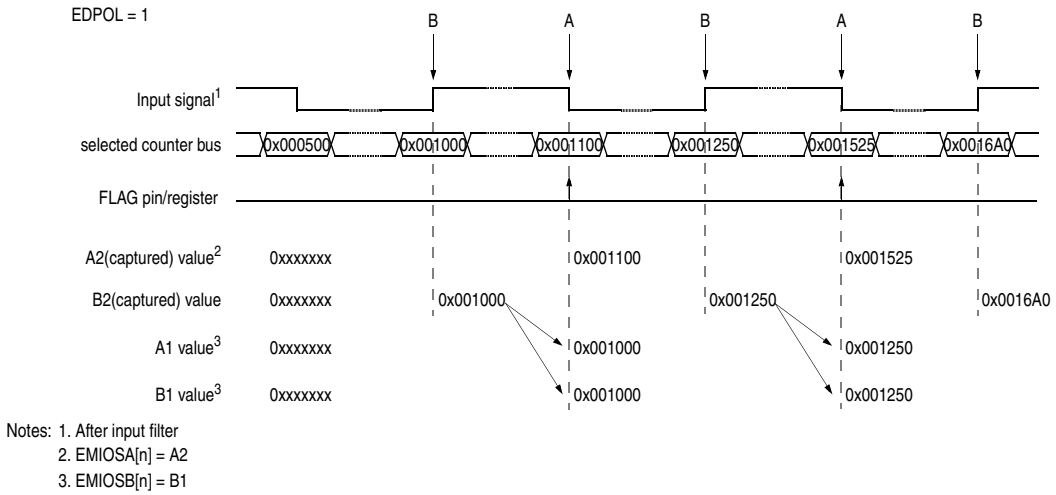


Figure 267. Input pulse width measurement example

Figure 268 shows the A1 and B1 updates when EMIOSA[n] and EMIOSB[n] register reads occur. Note that A1 register has always coherent data related to A2 register. Note also that when EMIOSA[n] read is performed B1 register is loaded with A1 register content. This guarantee that the data in register B1 has always the coherent data related to the last EMIOSA[n] read. The B1 register updates remains locked until EMIOSB[n] read occurs. If EMIOSA[n] read is performed B1 is updated with A1 register content even if B1 update is locked by a previous EMIOSA[n] read operation.

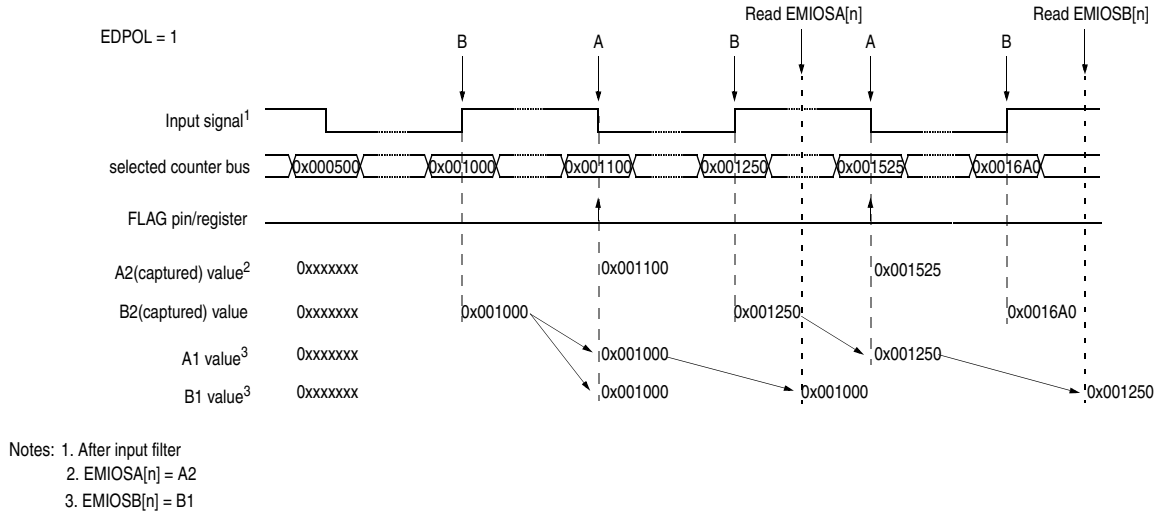


Figure 268. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads

Reading EMIOSA[n] followed by EMIOSB[n] always provides coherent data. If not coherent data is required for any reason, the sequence of reads should be inverted, therefore EMIOSB[n] should be read prior to EMIOSA[n] register. Note that even in this case B1 register updates will be blocked after EMIOSA[n] read, thus a second EMIOSB[n] is required in order to release B1 register updates.

Input Period Measurement (IPM) mode

The IPM mode (MODE[0:6] = 0000101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOSC[n] register.

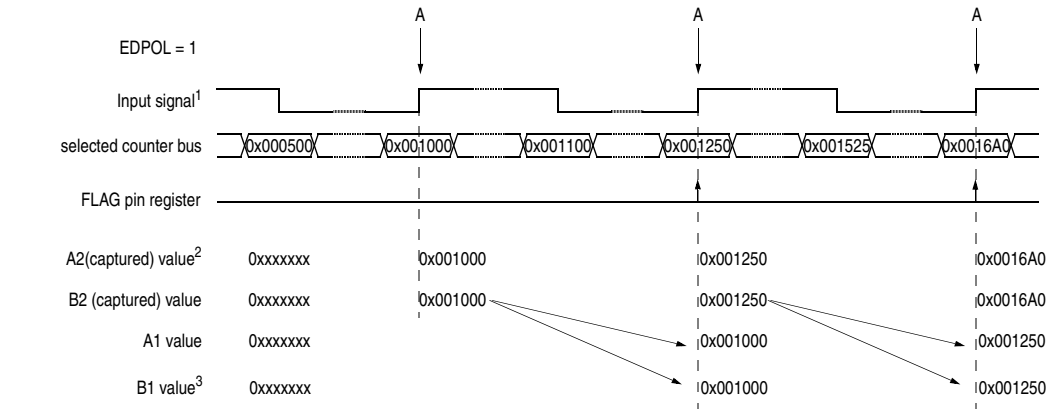
When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOSA[n] and EMIOSB[n] return the values in register A2 and B1, respectively.

In order to allow coherent data, reading EMIOSA[n] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOSB[n] register. Reading EMIOSB[n] register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

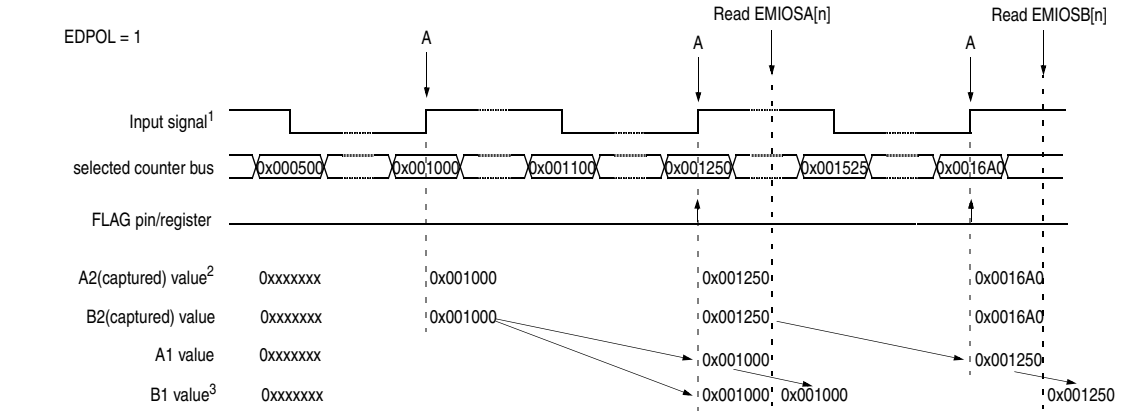
Figure 269 shows how the Unified Channel can be used for input period measurement.



Notes: 1. After input filter
2. EMIOSA[n] = A2
3. EMIOSB[n] = B1

Figure 269. Input period measurement example

Figure 270 describes the A1 and B1 register updates when EMIOSA[n] and EMIOSB[n] read operations are performed. When EMIOSA[n] read occurs the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOSB[n] is read. After EMIOSB[n] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the *Figure 270* example.



Notes: 1. After input filter
2. EMIOSA[n] = A2
3. EMIOSB[n] = B1

Figure 270. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads

Double Action Output Compare (DAOC) mode

In the DAOC mode the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B. There is no restriction concerning the order in which A and B matches occur.

When the DAOC mode is entered, coming out from GPIO mode both comparators are disabled and the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

Data written to A2 and B2 are transferred to A1 and B1, respectively, on the next system clock cycle if bit OU[n] of the EMIOSOUDIS register is cleared (see *Figure 273*). The transfer is blocked if bit OU[n] is set. Comparator A is enabled only after the transfer to A1 register occurs and is disabled on the next A match. Comparator B is enabled only after the transfer to B1 register occurs and is disabled on the next B match. Comparators A and B are enabled and disabled independently.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

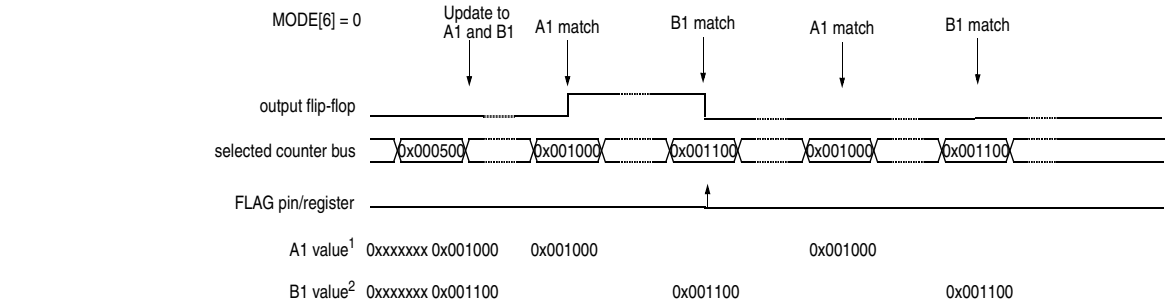
MODE[6] controls if the FLAG is set on both matches (MODE[0:6] = 0000111) or just on the B match (MODE[0:6] = 0000110). FLAG bit assertion depends on comparator enabling.

If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. Note that the FLAG bit is not affected by these forced operations.

Note: If both registers (A1 and B1) are loaded with the same value, the B match prevails concerning the output pin state (output flip-flop is set to the complement of EDPOL), the FLAG bit is set and both comparators are disabled.

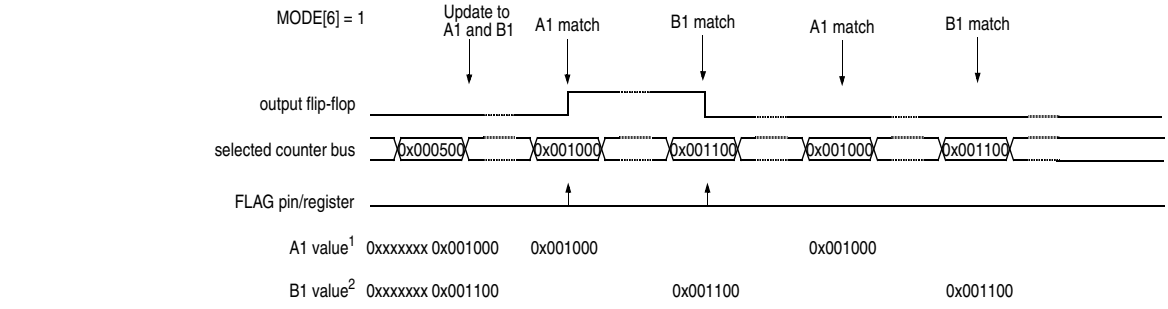
Figure 271 and *Figure 272* show how the Unified Channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



Notes: 1. EMIOSA[n] = A1 (when reading)
2. EMIOSB[n] = B1 (when reading)

A2 = A1according to OU[n] bit
B2 = B1according to OU[n] bit

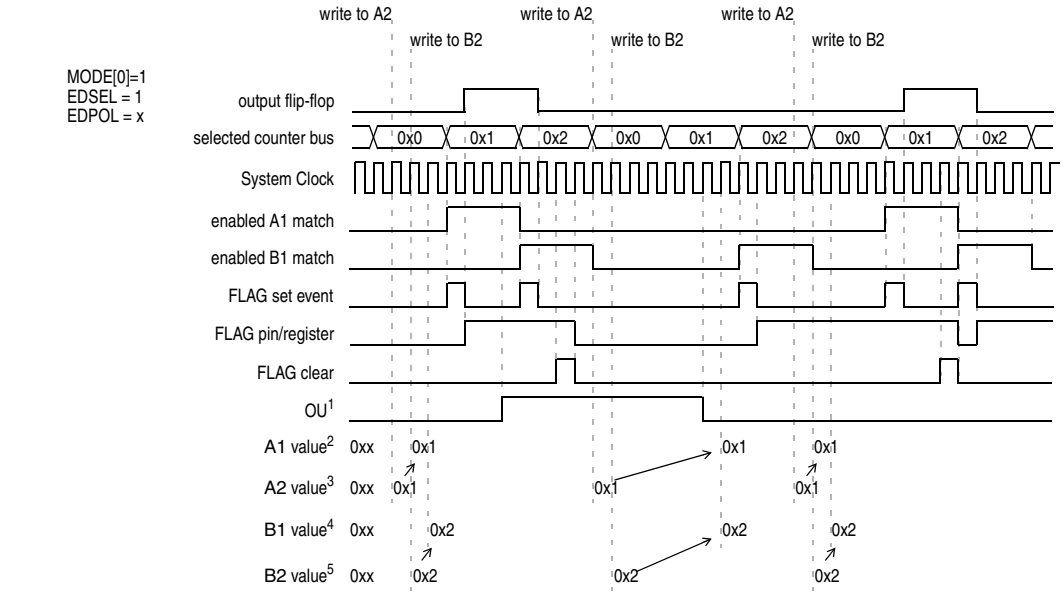
Figure 271. Double action output compare with FLAG set on the second match



Notes: 1. EMIOSA[n] = A1 (when reading)
2. EMIOSB[n] = B1 (when reading)

A2 = A1according to OU[n] bit
B2 = B1according to OU[n] bit

Figure 272. Double action output compare with FLAG set on both matches



Note: 1. OU[n] bit of EMIOSOUDIS register
2. EMIOSA[n] = A1 (when reading)
3. EMIOSA[n] = A2 (when writing)
4. EMIOSB[n] = B1 (when reading)
5. EMIOSB[n] = B2 (when writing)

Figure 273. DAOC with transfer disabling example

Modulus Counter (MC) mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

Bit MODE[6] selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the EMIOSC[n] register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. Bit MODE[4] selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter. The timing of those events varies according to the MC mode setup as follows:

- *Internal counter clearing on match start (MODE[0:6] = 001000b)*
  - External clock is selected if MODE[6] is set. In this case the internal counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. Note that by having the internal counter cleared as soon as the match occurs and incremented at the next input event a shorter zero count is generated. See *Figure 296* and *Figure 297*.
  - Internal clock source is selected if MODE[6] is cleared. In this case the counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. At the next prescaler tick after the match the internal

counter remains at zero and only resumes counting on the following tick. See *Figure 296* and *Figure 298*.

- *Internal counter clearing on match end (MODE[0:6] = 001001b)*
  - External clock is selected if MODE[6] is set. In this case the internal counter clears when the match signal is asserted and the input event occurs. The channel FLAG is set at the same time the counter is cleared. See *Figure 296* and *Figure 299*.
  - Internal clock source is selected if MODE[6] is cleared. In this case the internal counter clears when the match signal is asserted and the prescaler tick occurs. The channel FLAG is set at the same time the counter is cleared. See *Figure 296* and *Figure 299*.

Note: If the internal clock source is selected and the prescaler of the internal counter is set to '1', the MC mode behaves the same way even in Clear on Match Start or Clear on Match End submodes.

When in up/down count mode (MODE[0:6] = 00101bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

Only values different than 0x0 must be written at A register. Loading 0x0 leads to unpredictable results.

Updates on A register or counter in MC mode may cause loss of match in the current cycle if the transfer occurs near the match. In this case, the counter may rollover and resume operation in the next cycle.

Register B2 has no effect in MC mode. Nevertheless, register B2 can be accessed for reads and writes by addressing EMIOSB.

Figure 274 and *Figure 275* show how the Unified Channel can be used as modulus counter in up mode and up/down mode, respectively.

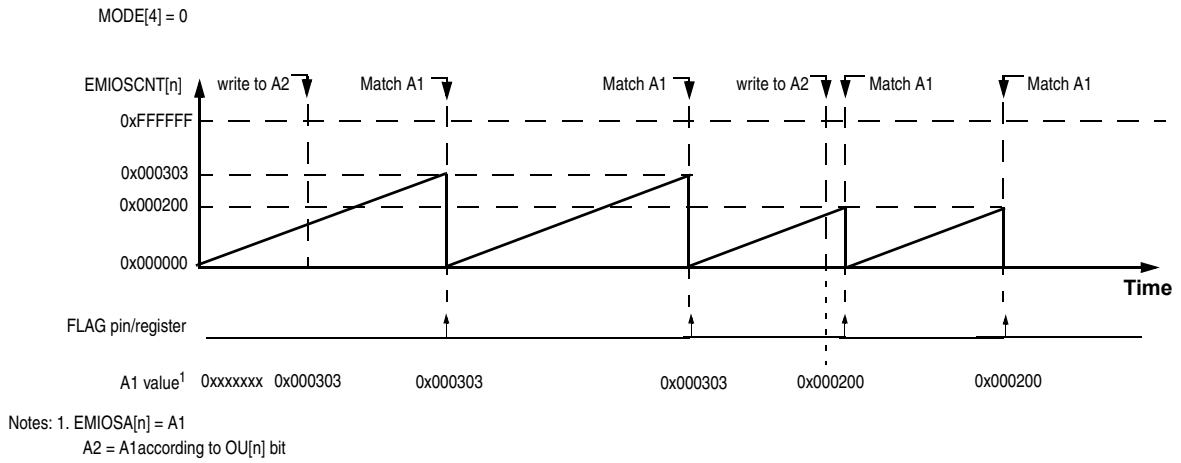
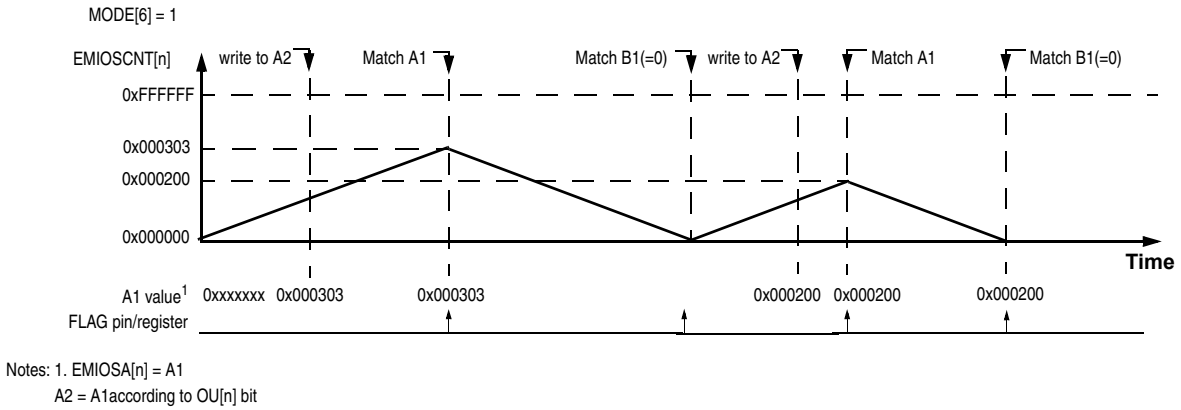


Figure 274. Modulus Counter Up mode example



Notes: 1. EMIOSA[n] = A1
A2 = A1according to OU[n] bit

Figure 275. Modulus Counter Up/Down mode example

Modulus Counter Buffered (MCB) mode

The MCB mode provides a time base which can be shared with other channels through the internal counter buses. Register A1 is double buffered thus allowing smooth transitions between cycles when changing A2 register value on the fly. A1 register is updated at the cycle boundary, which is defined as when the internal counter transitions to 0x1.

The internal counter values operates within a range from 0x1 up to register A1 value. If when entering MCB mode coming out from GPIO mode the internal counter value is not within that range then the A match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xFFFF for a 16-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal MCB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to A1 register value range when the MCB mode is entered.

Bit MODE[6] selects internal clock source if cleared or external if set. When external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOSC[n] channel register.

When entering in MCB mode, if up counter is selected by MODE[4] = 0 (MODE[0:6] = 101000b), the internal counter starts counting from its current value to up direction until A1 match occurs. The internal counter is set to 0x1 when its value matches A1 value and a clock tick occurs (either prescaled clock or input pin event).

If up/down counter is selected by setting MODE[4] = 1, the counter changes direction at A1 match and counts down until it reaches the value 0x1. After it has reached 0x1 it is set to count in up direction again. B1 register is used to generate a match in order to set the internal counter in up-count direction if up/down mode is selected. Register B1 cannot be changed while this mode is selected.

Note that differently from the MC mode, the MCB mode counts between 0x1 and A1 register value. Only values greater than 0x1 must be written at A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression: (2*A1)-2.

Figure 276 describes the counter cycle for several A1 values. Register A1 is loaded with A2 register value at the cycle boundary. Thus any value written to A2 register within cycle **n** will be updated to A1 at the next cycle boundary and therefore will be used on cycle **n+1**. The

cycle boundary between cycle **n** and cycle **n+1** is defined as when the internal counter transitions from A1 value in cycle **n** to 0x1 in cycle **n+1**. Note that the FLAG is generated at the cycle boundary and has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

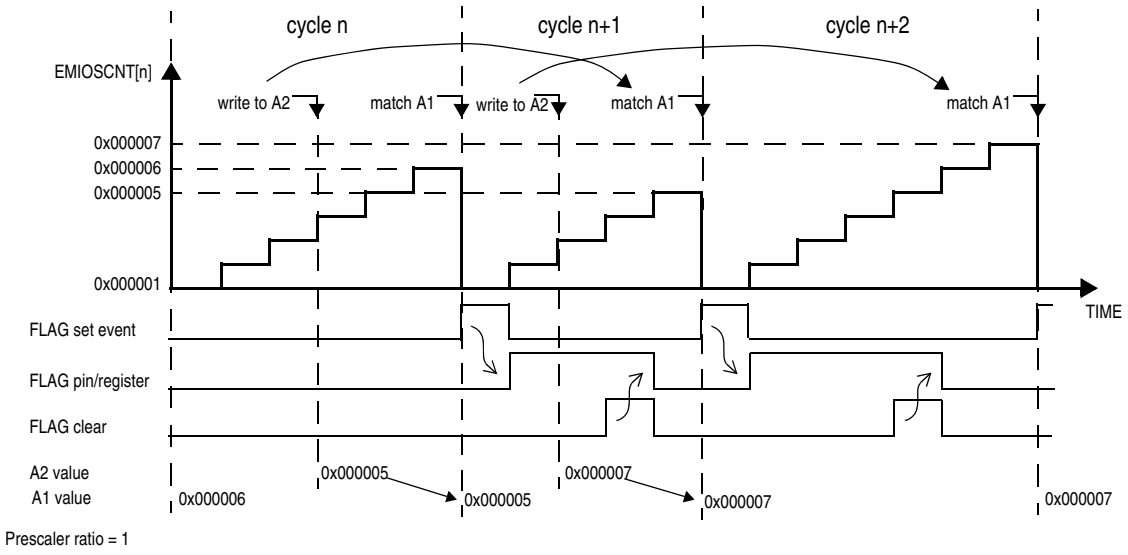


Figure 276. Modulus Counter Buffered (MCB) Up Count mode

Figure 277 describes the MCB in up/down counter mode (MODE[0:6] = 10101bb). A1 register is updated at the cycle boundary. If A2 is written in cycle **n**, this new value will be used in cycle **n+1** for A1 match. Flags are generated only at A1 match start if MODE[5] is 0. If MODE[5] is set to 1 flags are also generated at the cycle boundary.

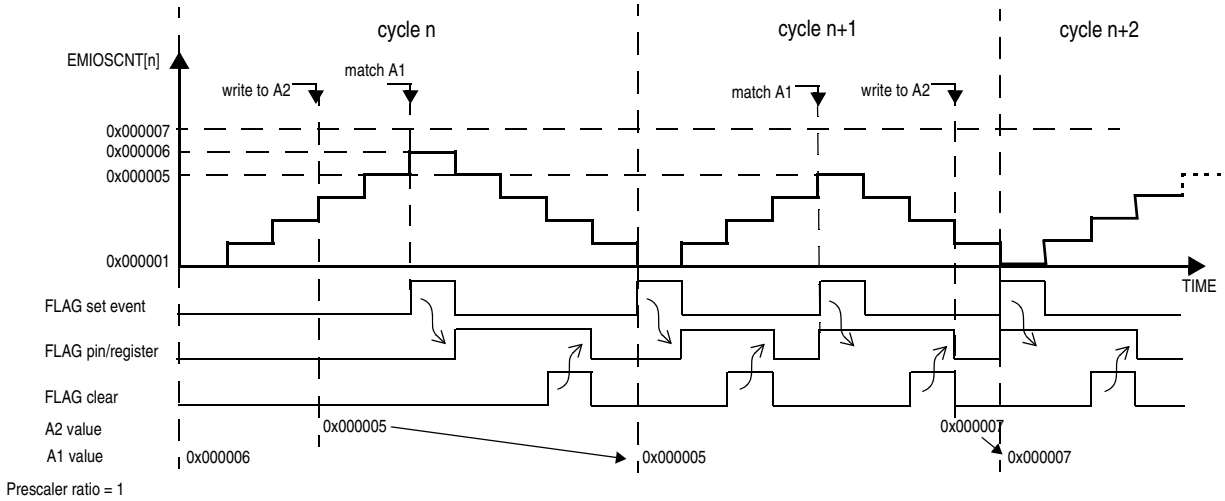


Figure 277. Modulus Counter Buffered (MCB) Up/Down mode

Figure 278 describes in more detail the A1 register update process in up counter mode. The A1 load signal is generated at the last system clock period of a counter cycle. Thus, A1 is updated with A2 value at the same time that the counter (EMIOSCNT[n]) is loaded with 0x1.

The load signal pulse has the duration of one system clock period. If A2 is written within cycle **n** its value is available at A1 at the first clock of cycle **n+1** and the new value is used for match at cycle **n+1**. The update disable bits OU[n] of EMIOSOUDIS register can be used to control the update of this register, thus allowing to delay the A1 register update for synchronization purposes.

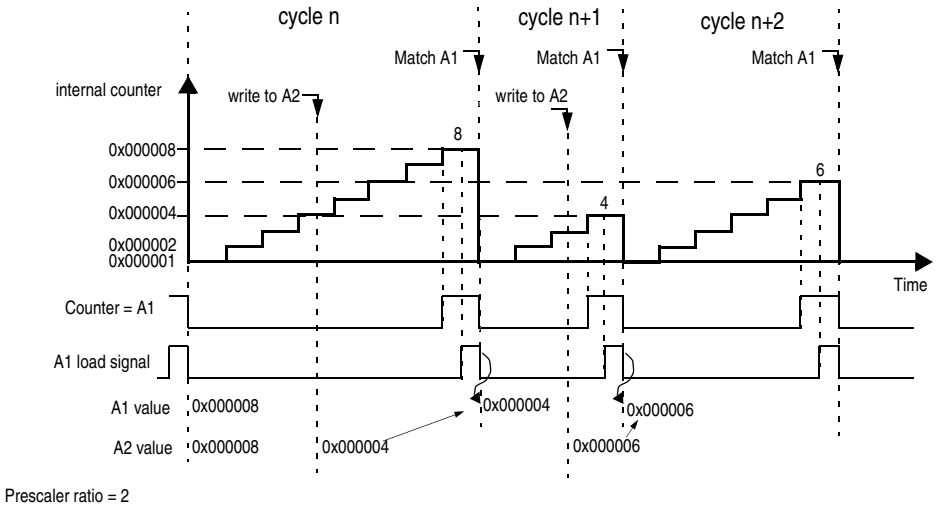


Figure 278. MCB Mode A1 Register Update in Up Counter mode

Figure 279 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle **n** in order to be used in cycle **n+1**. Thus A1 receives this new value at the next cycle boundary. Note that the update disable bits OU[n] of EMIOSOUDIS register can be used to disable the update of A1 register.

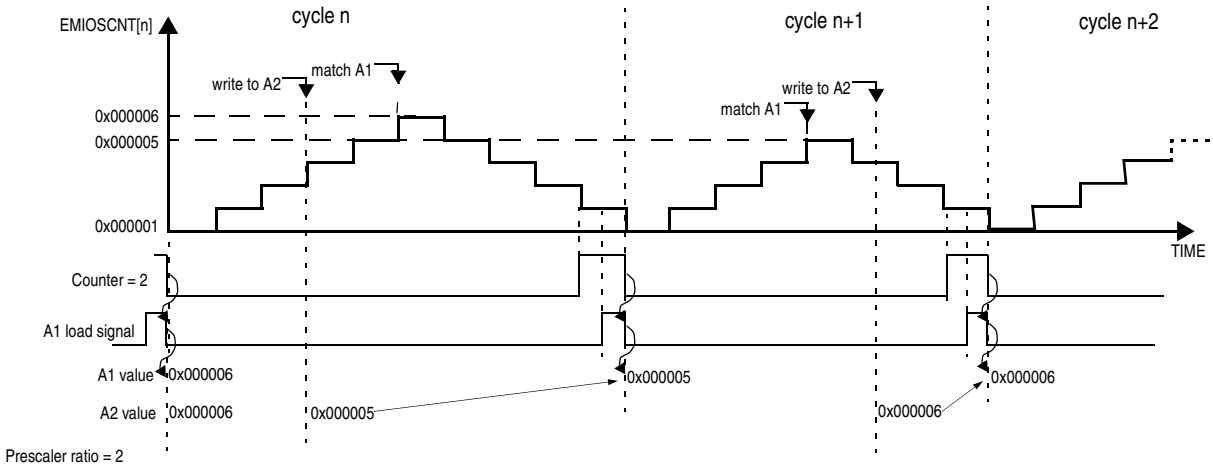


Figure 279. MCB Mode A1 Register Update in Up/Down Counter mode

Output Pulse Width and Frequency Modulation Buffered (OPWFMB) mode

This mode (MODE[0:6] = 10110b0) provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this

mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values on the fly. 0% and 100% duty cycles are supported.

At OPWFMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOSC[n] register.

If when entering OPWFMB mode coming out from GPIO mode the internal counter value is not within that range then the B match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xFFFF for a 16-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal OPWFMB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to B1 register value range when the OPWFMB mode is entered.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 0x1, thus restarting the counter cycle.

Only values greater than 0x1 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results. If you want to configure the module for OPWFMB mode, ensure that the B1 register is modified before the mode is set.

Figure 280 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. Note that the output pin transition occurs when the A1 or B1 match signal is deasserted which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x4 the output pin transitions 4 counter periods after the cycle had started, plus one system clock cycle. Note that in the example shown in *Figure 280* the internal counter prescaler has a ratio of two.

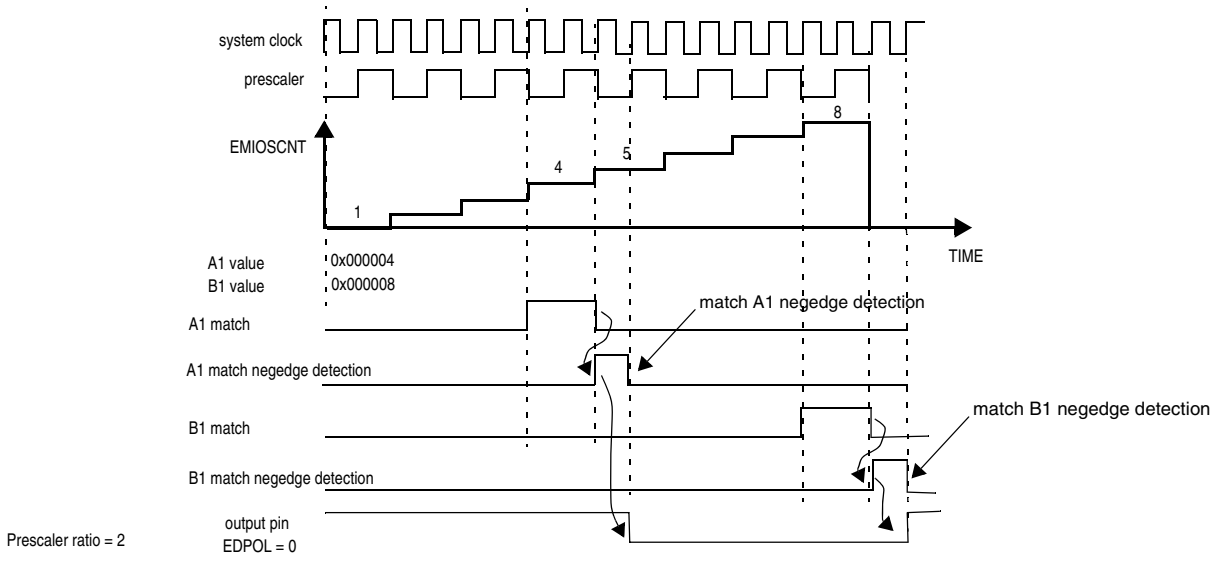


Figure 280. OPWFMB A1 and B1 match to Output Register Delay

Figure 281 describes the generated output signal if A1 is set to 0x0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1 = 0x1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin

transition instead of the negedge used when A1 = 0x1. Note that A1 posedge match signal from cycle **n+1** occurs at the same time as B1 negedge match signal from cycle **n**. This allows to use the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

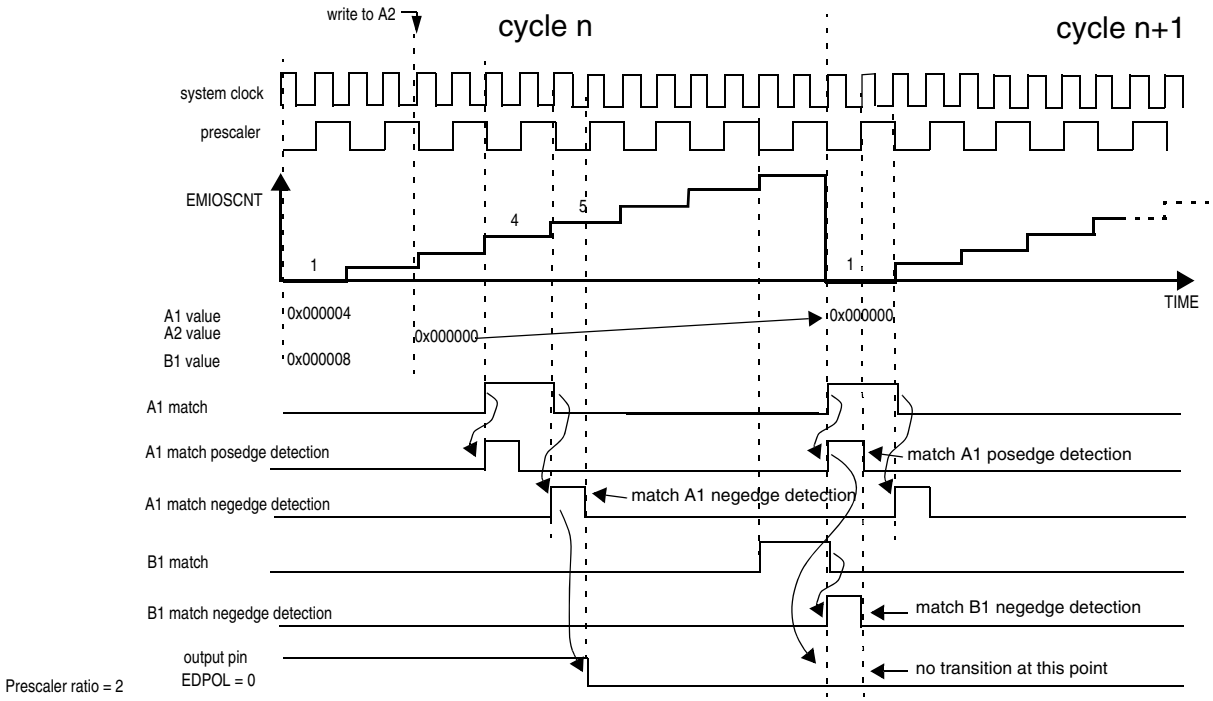


Figure 281. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 282 describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated at the last system clock period of a counter cycle. Thus, A1 and B1 are updated respectively with A2 and B2 values at the same time that the counter (EMIOSCNT[n]) is loaded with 0x1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock period. If A2 and B2 are written within cycle **n** their values are available at A1 and B1, respectively, at the first clock of cycle **n+1** and the new values are used for matches at cycle **n+1**. The update disable bits OU[n] of EMIOSOUDIS register can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In *Figure 282* it is assumed that both the channel and global prescalers are set to 0x1 (each divide ratio is two), meaning that the channel internal counter transitions at every four system clock cycles. FLAGs can be generated only on B1 matches when MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. Since B1 flag occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle **n** were loaded to A1 or B1, respectively, thus generating matches in cycle **n+1**. Note that the FLAG has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

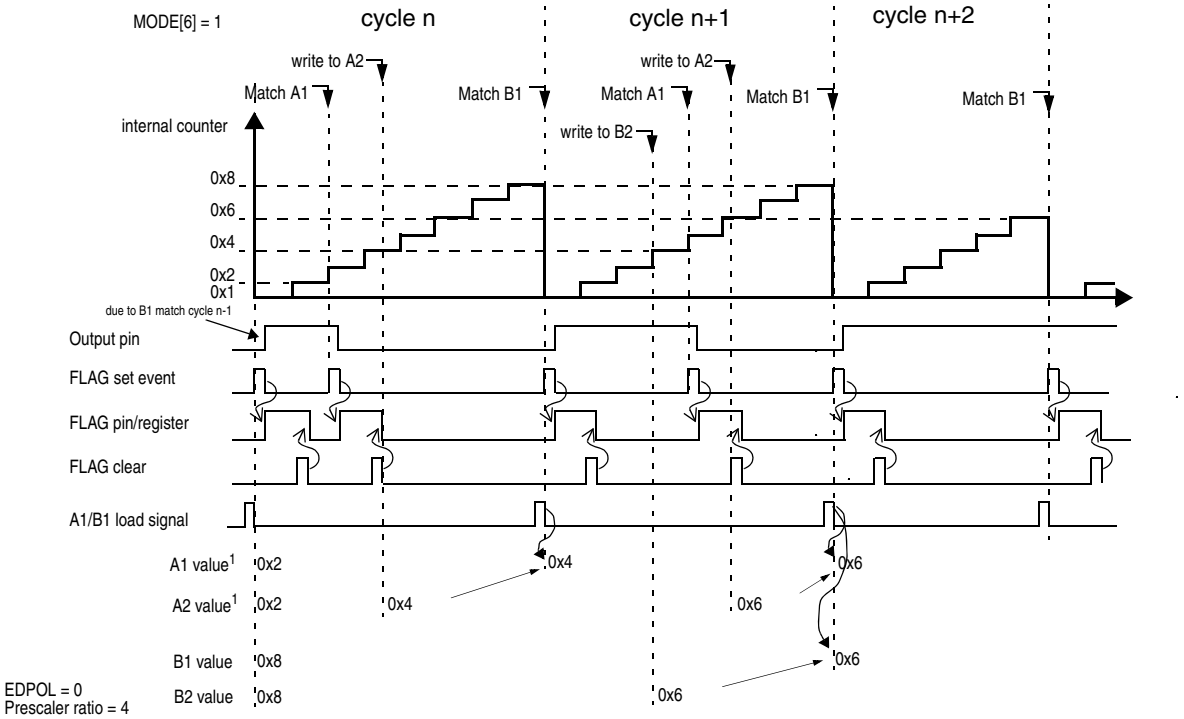


Figure 282. OPWFMB A1 and B1 registers update and flags

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similarly to a B1 match FORCMB sets the internal counter to 0x1. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 283 describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL = 0 and the resultant prescaler value is 1. Initially A1 = 0x8 and B1 = 0x8. In this case, B1 match has precedence over A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

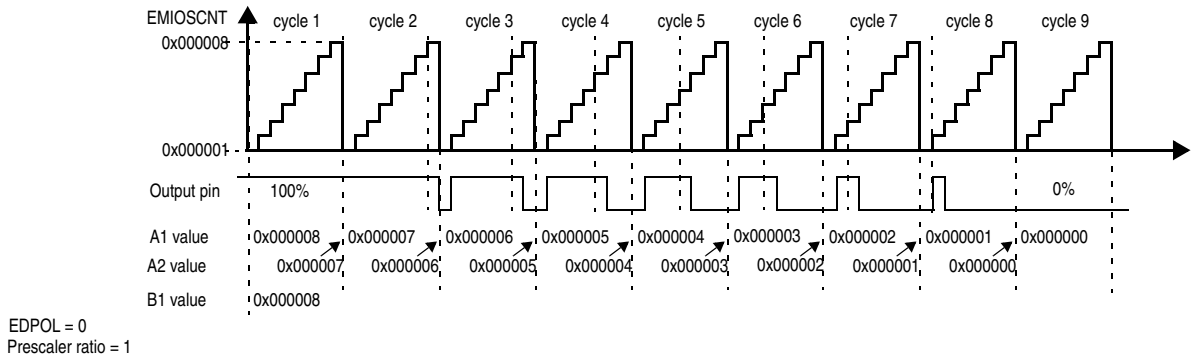


Figure 283. OPWFMB mode from 100% to 0% duty cycle

A 0% duty cycle signal is generated if A1 = 0x0 as shown in *Figure 283* cycle 9. In this case B1 = 0x8 match from cycle 8 occurs at the same time as the A1 = 0x0 match from cycle 9.

Please, refer to *Figure 281* for a description of the A1 and B1 match generation. In this case A1 match has precedence over B1 match and the output signal transitions to EDPOL.

Center Aligned Output PWM Buffered with Dead-Time (OPWMCB) mode

This operation mode generates a center aligned PWM with dead time insertion to the leading (MODE[0:6] = 10111b1) or trailing edge (MODE[0:6] = 10111b0). A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values on the fly.

Bits BSL[0:1] select the time base. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB Up/Down mode, as shown in *Figure 277*. It is recommended to start the MCB channel time base after the OPWMCB mode is entered in order to avoid missing A matches at the very first duty cycle.

Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base.

Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Bit Mode[6] selects between trailing and leading dead time insertion, respectively.

Note: The internal counter runs in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio.

When OPWMCB mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

The following basic steps summarize proper OPWMCB startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable Global Prescaler;
2. *[MCB channel]* Disable Channel Prescaler;
3. *[MCB channel]* Write 0x1 at internal counter;
4. *[MCB channel]* Set A register;
5. *[MCB channel]* Set channel to MCB Up mode;
6. *[MCB channel]* Set prescaler ratio;
7. *[MCB channel]* Enable Channel Prescaler;
8. *[OPWMCB channel]* Disable Channel Prescaler;
9. *[OPWMCB channel]* Set A register;
10. *[OPWMCB channel]* Set B register;
11. *[OPWMCB channel]* Select time base input through BSL[1:0] bits;
12. *[OPWMCB channel]* Enter OPWMCB mode;
13. *[OPWMCB channel]* Set prescaler ratio;
14. *[OPWMCB channel]* Enable Channel Prescaler;
15. *[global]* Enable Global Prescaler.

Figure 284 describes the load of A1 and B1 registers which occurs when the selected counter bus transitions from 0x2 to 0x1. This event defines the cycle boundary. Note that values written to A2 or B2 within cycle **n** are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle **n+1**.

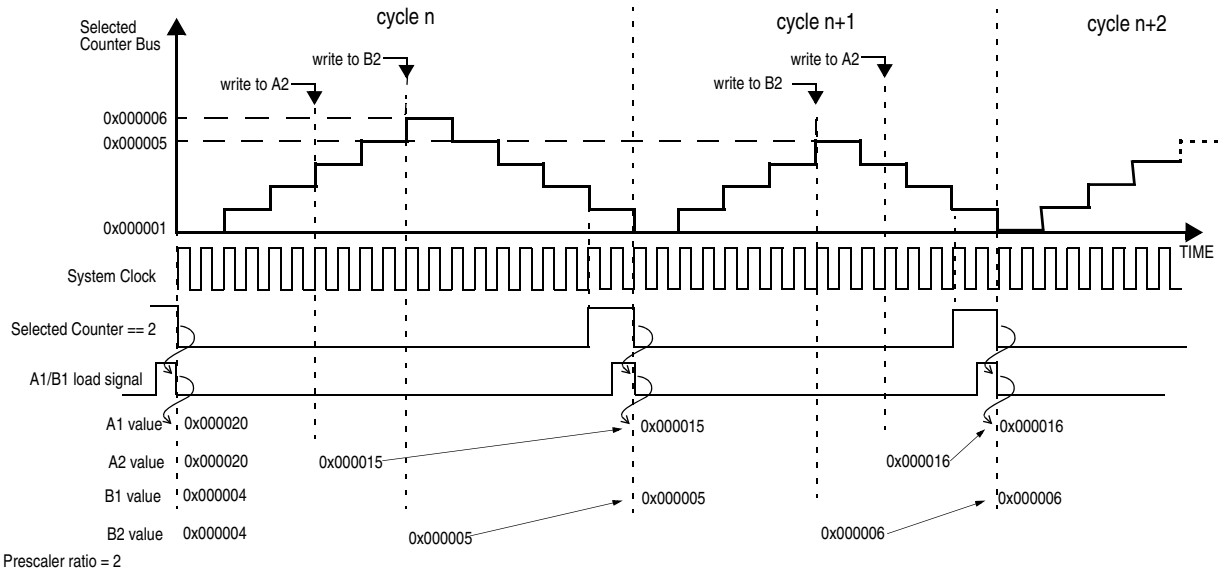


Figure 284. OPWMCB A1 and B1 registers load

Bit OU[n] of the EMIOSOUDIS register can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Note that using the update disable bit A1 and B1 registers can be updated at the same counter cycle thus allowing to change both registers at the same time.

In this mode A1 matches always sets the internal counter to 0x1. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x1. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x0 as consequence of a rollover. In order to avoid it the user should not write to the EMIOSB register a value greater than twice the difference between external count up limit and EMIOSA value.

Figure 285 shows two cycles of a Center Aligned PWM signal. Note that both A1 and B1 register values are changing within the same cycle which allows to vary at the same time the duty cycle and dead time values.

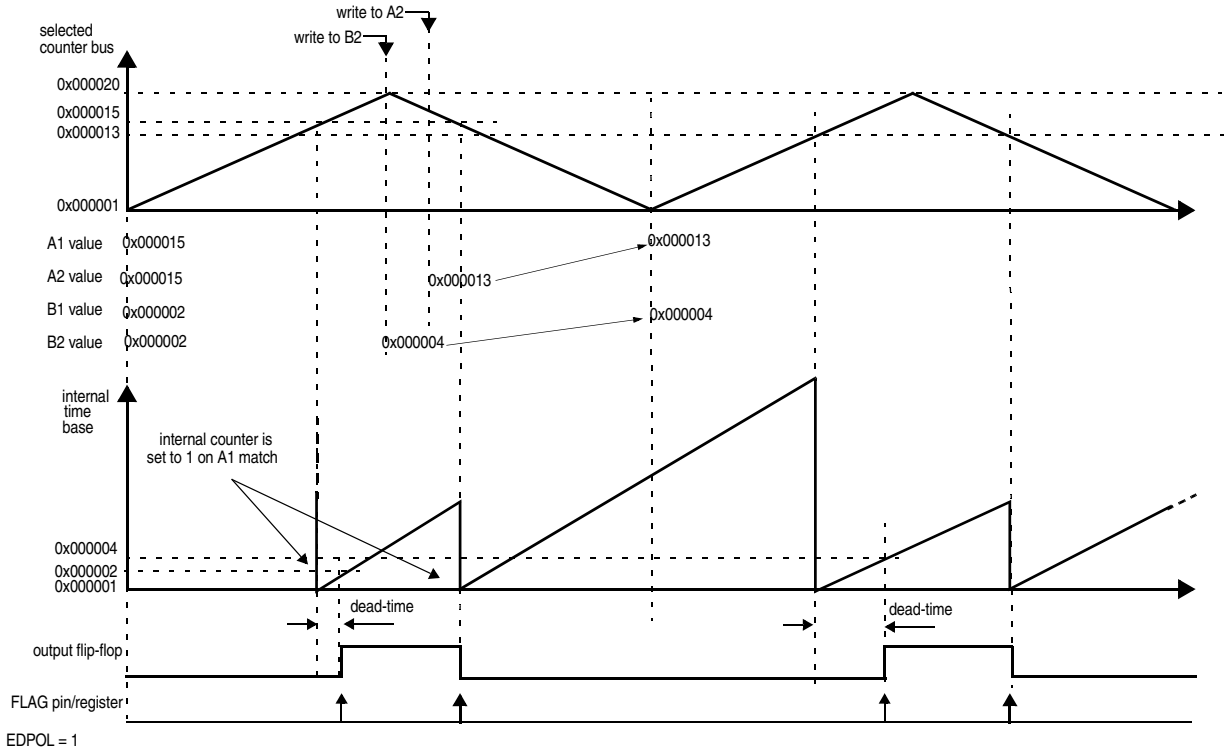


Figure 285. OPWMCB with lead dead time insertion

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x1. In the second match between register A1 and the selected time base, the internal counter is set to 0x1 and B1 matches are enabled. When the match between register B1 and the selected time base occurs the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

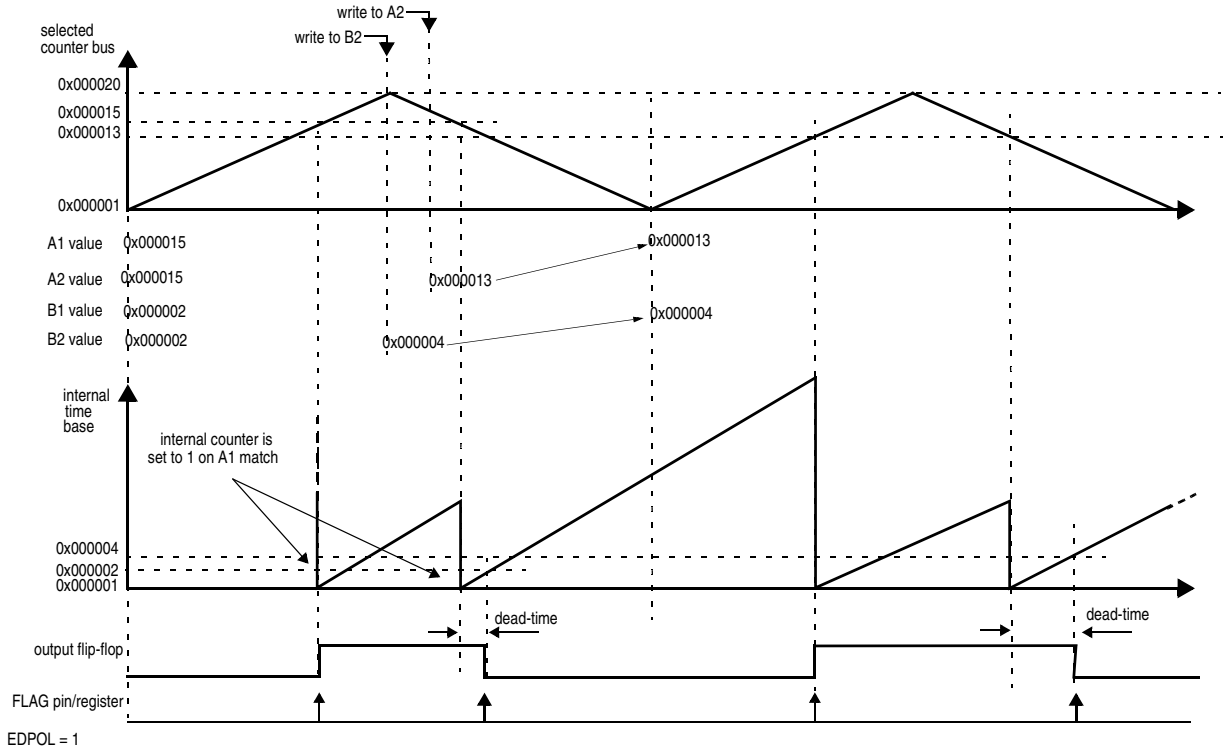


Figure 286. OPWMCB with trail dead time insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

Note: In OPWMCB mode, FORCMA and FORCMB do not have the same behavior as a regular match. Instead, they force the output flip-flop to constant value which depends upon the selected dead time insertion mode, lead or trail, and the value of the EDPOL bit.

FORCMA has different behaviors depending upon the selected dead time insertion mode, lead or trail. In lead dead time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead time insertion the output flip-flop is forced to the value of EDPOL bit.

If bit FORCMB is set, the output flip-flop value depends upon the selected dead time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

Note: FORCMA bit set does not set the internal time-base to 0x1 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

Note: FORCMA and FORCMB have the same behavior even in Freeze or normal mode regarding the output pin transition.

When FORCMA is issued along with FORCMB the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that.FORCMA has precedence over FORCMB

when lead dead time insertion is selected and FORCMB has precedence over FORCMA when trail dead time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting A1 = 1 generates a 100% duty cycle waveform. Assuming EDPOL is set to '1' and OPWMCB mode with trail dead time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1). If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced, only if the pin starts the current cycle in the opposite of EDPOL value. In case of 100% duty cycle, the transition from EDPOL to the opposite of EDPOL may be obtained by forcing pin, using FORCMA or FORCMB, or both.

Note: If A1 is set to 0x1 at OPWMCB entry the 100% duty cycle may not be obtained in the very first PWM cycle due to the pin condition at mode entry.

Only values different than 0x0 are allowed to be written to A1 register. If 0x0 is loaded to A1 the results are unpredictable.

Note: A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead time insertion B1 match from cycle **n** could eventually cross the cycle boundary and occur in cycle **n+1**. In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle **n+1** are not affected by the late B1 matches from cycle **n**.

Figure 287 shows a 100% duty cycle output signal generated by setting A1 = 4 and B1 = 3. In this case the trailing edge is positioned at the boundary of cycle **n+1**, which is actually considered to belong to cycle **n+2** and therefore does not cause the output flip-flip to transition.

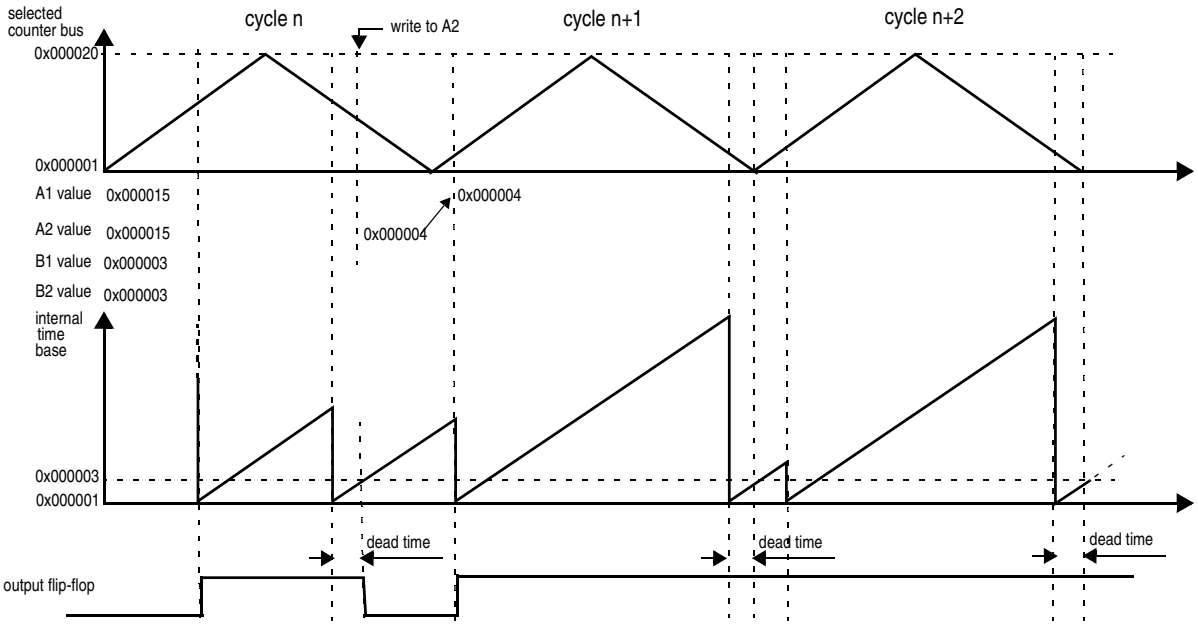


Figure 287. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3)

It is important to notice that, such as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal which compares the selected time base with A1 or B1 register values. Please refer to *Figure 280* which describes the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

Output Pulse Width Modulation Buffered (OPWMB) Mode

OPWMB mode (MODE[0:6] = 11000b0) is used to generate pulses with programmable leading and trailing edge placement. An external counter driven in MCB Up mode must be selected from one of the counter buses. A1 register value defines the first edge and B1 the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please refer to *Figure 282* for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or in both A1 and B1 matches, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG bit is not set by the FORCMA and FORCMB operations.

At OPWMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOSC[n] register.

Some rules applicable to the OPWMB mode are:

- *B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle*
- *A1 = 0 match from cycle **n** has precedence over B1 match from cycle **n-1***
- *A1 matches are masked out if they occur after B1 match within the same cycle*
- *Any value written to A2 or B2 on cycle **n** is loaded to A1 and B1 registers at the following cycle boundary (assuming OU[n] bit of EMIOSOUDIS register is not asserted). Thus the new values will be used for A1 and B1 matches in cycle **n+1***

Figure 288 describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to '0'.

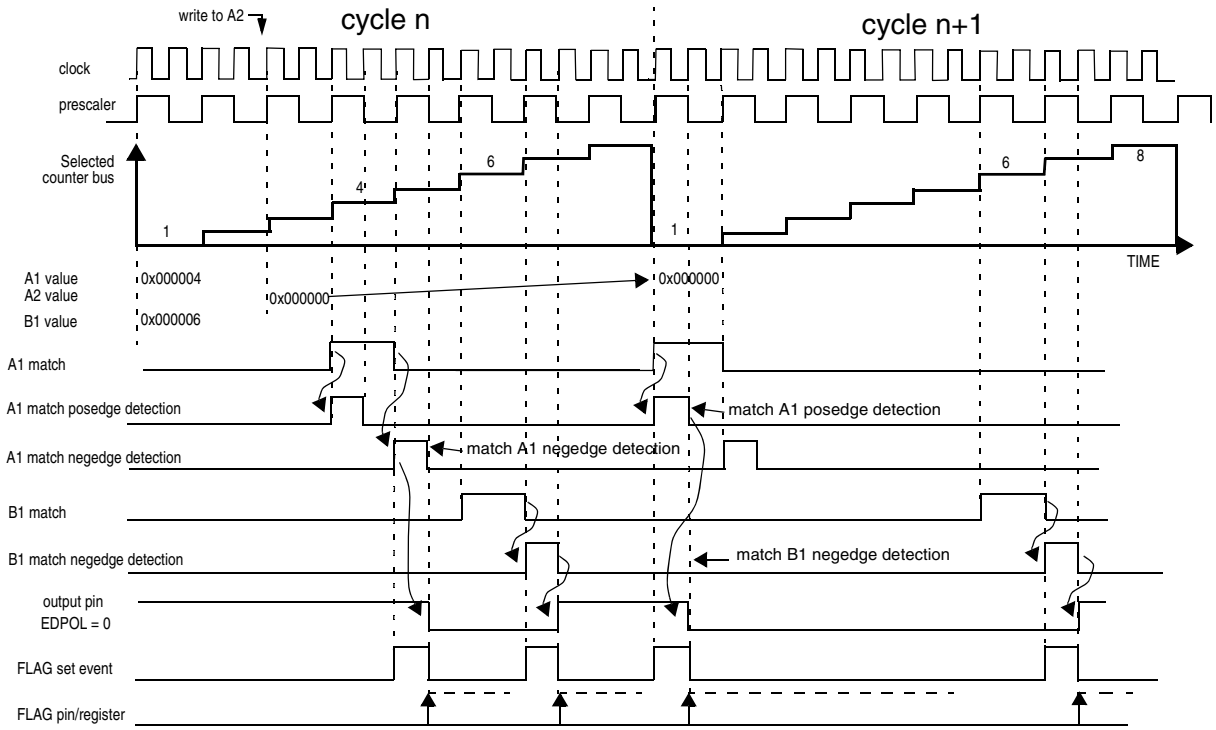


Figure 288. OPWMB mode matches and flags

Note that the output pin transitions are based on the negedges of the A1 and B1 match signals. *Figure 288* shows in cycle **n+1** the value of A1 register being set to '0'. In this case the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 289 describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1 = 0x8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

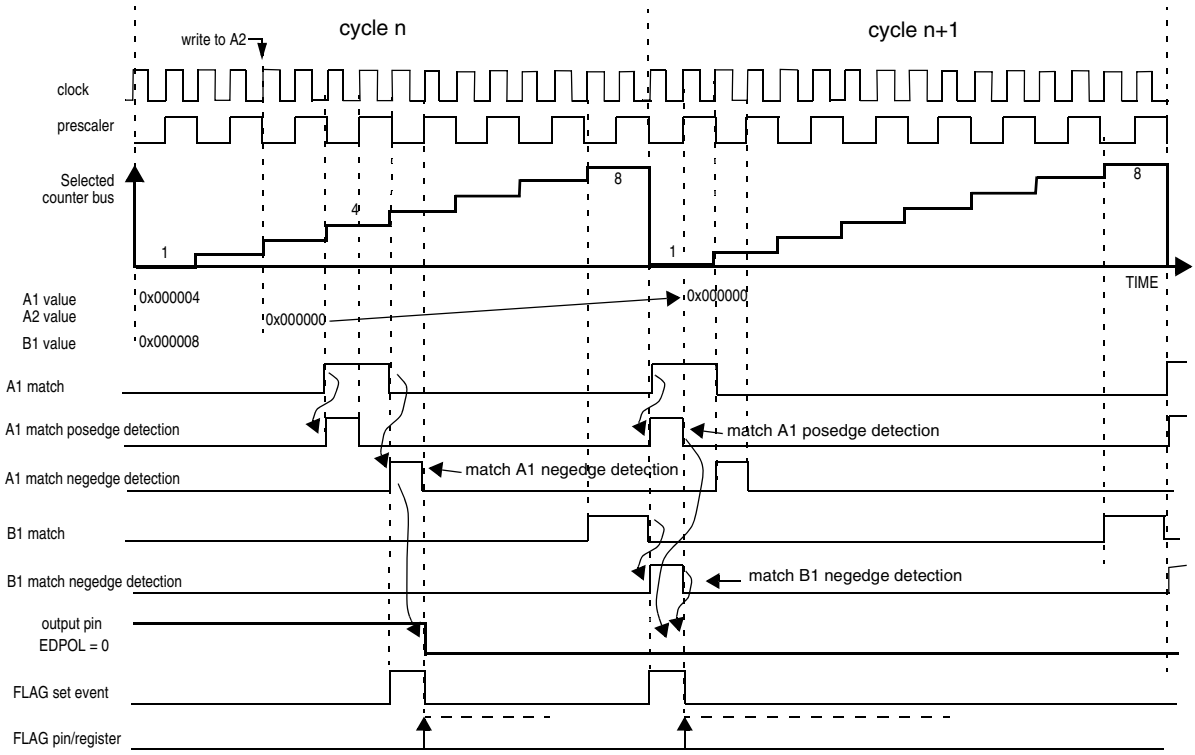


Figure 289. OPWMB mode with 0% duty cycle

Figure 290 shows a waveform changing from 100% to 0% duty cycle. EDPOL in this case is zero. In this example B1 is programmed to the same value as the period of the external selected time base.

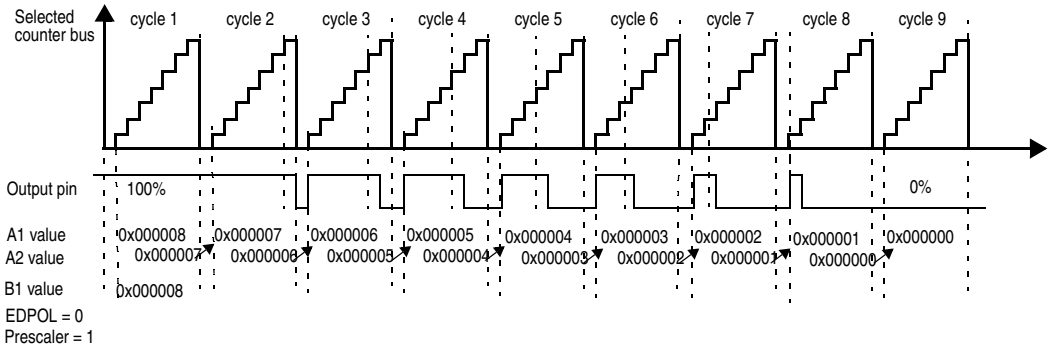


Figure 290. OPWMB mode from 100% to 0% duty cycle

In *Figure 290* if B1 is set to a value lower than 0x8 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. Note also that if B1 is set to 0x9, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

Output Pulse Width Modulation with Trigger (OPWMT) mode

OPWMT mode (MODE[0:6] = 0100110) is intended to support the generation of pulse width modulation signals where the period is not modified while the signal is being output, but where the duty cycle will be varied and must not create glitches. The mode is intended to be used in conjunction with other channels executing in the same mode and sharing a common timebase. It will support each channel with a fixed PWM leading edge position with respect to the other channels and the ability to generate a trigger signal at any point in the period that can be output from the module to initiate activity in other parts of the device such as starting ADC conversions.

An external counter driven in either MC Up or MCB Up mode must be selected from one of the counter buses.

Register A1 defines the leading edge of the PWM output pulse and as such the beginning of the PWM's period. This makes it possible to insure that the leading edge of multiple channels in OPWMT mode can occur at a specific time with respect to the other channels when using a shared timebase. This can allow the introduction of a fixed offset for each channel which can be particularly useful in the generation of lighting PWM control signals where it is desirable that edges are not coincident with each other to help eliminate noise generation. The value of register A1 represents the shift of the PWM channel with respect to the selected timebase. A1 can be configured with any value within the range of the selected time base. Note that registers loaded with 0x0 will not produce matches if the timebase is driven by a channel in MCB mode.

A1 is not buffered as the shift of a PWM channel must not be modified while the PWM signal is being generated. In case A1 is modified it is immediately updated and one PWM pulse could be lost.

EMIOSB[n] address gives access to B2 register for write and B1 register for read. Register B1 defines the trailing edge of the PWM output pulse and as such the duty cycle of the PWM signal. To synchronize B1 update with the PWM signal and so ensure a correct output pulse generation the transfer from B2 to B1 is done at every match of register A1.

EMIOSOUDIS register affects transfers between B2 and B1 only.

In order to account for the shift in the leading edge of the waveform defined by register A1 it will be necessary that the trailing edge, held in register B1, can roll over into the next period. This means that a match against the B1 register should not have to be qualified by a match in the A1 register. The impact of this would mean that incorrectly setting register B1 to a value less that register A1 will result in the output being held over a cycle boundary until the B1 value is encountered.

This mode provides a buffered update of the trailing edge by updating register B1 with register B2 contents only at a match of register A1.

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A1 occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

Note that the output pin and flag transitions are based on the posedges of the A1, B1 and A2 match signals. Please, refer to *Figure 288* at *Section , Output Pulse Width Modulation Buffered (OPWMB) Mode* for details on match posedge.

Register A2 defines the generation of a trigger event within the PWM period and A2 should be configured with any value within the range of the selected time base, otherwise no trigger

will be generated. A match on the comparator will generate the FLAG signal but it has no effect on the PWM output signal generation. The typical setup to obtain a trigger with FLAG is to enable DMA and to drive the channel's ipd_done input high.

A2 is not buffered and therefore its update is immediate. If the channel is running when a change is made this could cause either the loss of one trigger event or the generation of two trigger events within the same period. Register A2 can be accessed by reading or writing the eMIOS UC Alternate A Register (EMIOSALTA) at UC[n] base address +0x14.

FLAG signal is set only at match on the comparator with A2. A match on the comparator with A1 or B1 or B2 has no effect on FLAG.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Any FORCMA and/or FORCMB has priority over any simultaneous match regarding to output pin transitions. Note that the load of B2 content on B1 register at an A match is not inhibited due to a simultaneous FORCMA/FORCMB assertion. If both FORCMA and FORCMB are asserted simultaneously the output pin goes to the opposite of EDPOL value such as if A1 and B1 registers had the same value. FORCMA assertion causes the transfer from register B2 to B1 such as a regular A match, regardless of FORCMB assertion.

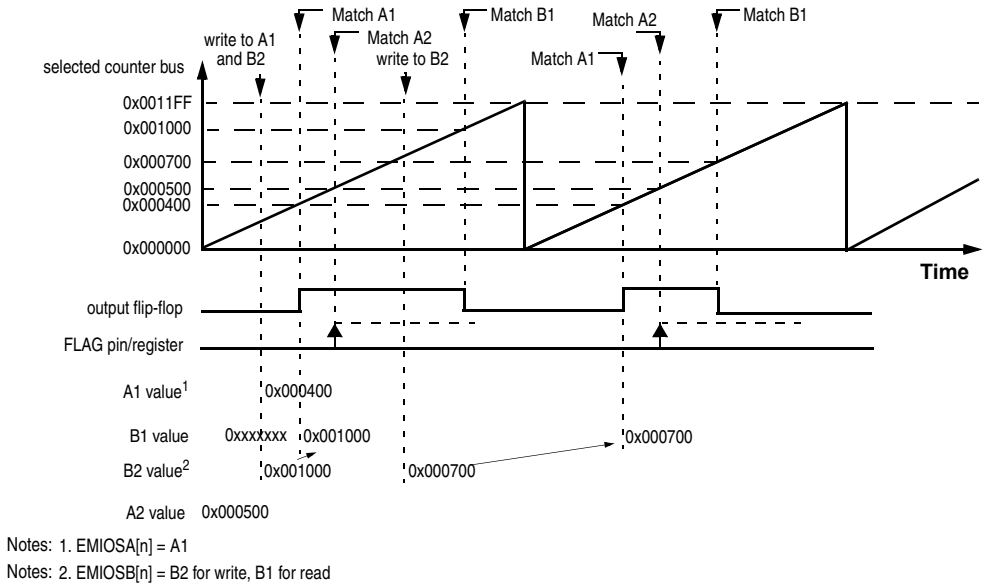
If subsequent matches occur on comparators A1 and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

At OPWMT mode entry the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

In order to achieve 0% duty cycle both registers A1 and B must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the complement value of EDPOL.

In order to achieve 100% duty cycle the register B1 must be set to a value greater than maximum value of the selected time base. As a consequence, if 100% duty cycle must be implemented, the maximum counter value for the time base is 0xFFFE for a 16-bit counter. When a match on comparator A1 occurs the output flip-flop is set at every period to the value of EDPOL bit. The transfer from register B2 to B1 is still triggered by the match at comparator A.

Figure 291 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and duty cycle update on next period update.

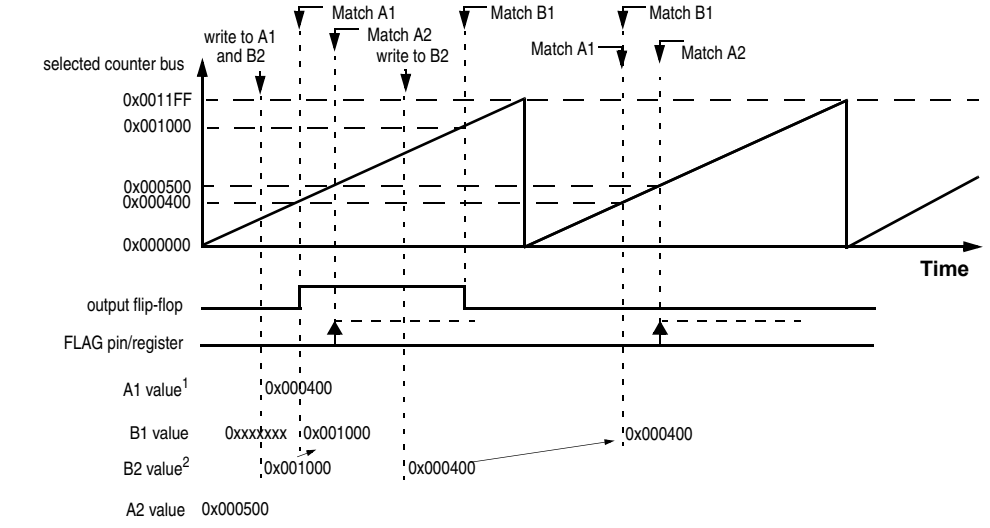


Notes: 1. EMIOSA[n] = A1

Notes: 2. EMIOSB[n] = B2 for write, B1 for read

Figure 291. OPWMT example

Figure 292 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 0% duty.



Notes: 1. EMIOSA[n] = A1

Notes: 2. EMIOSB[n] = B2 for write, B1 for read

Figure 292. OPWMT with 0% Duty Cycle

Figure 293 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 100% duty cycle.

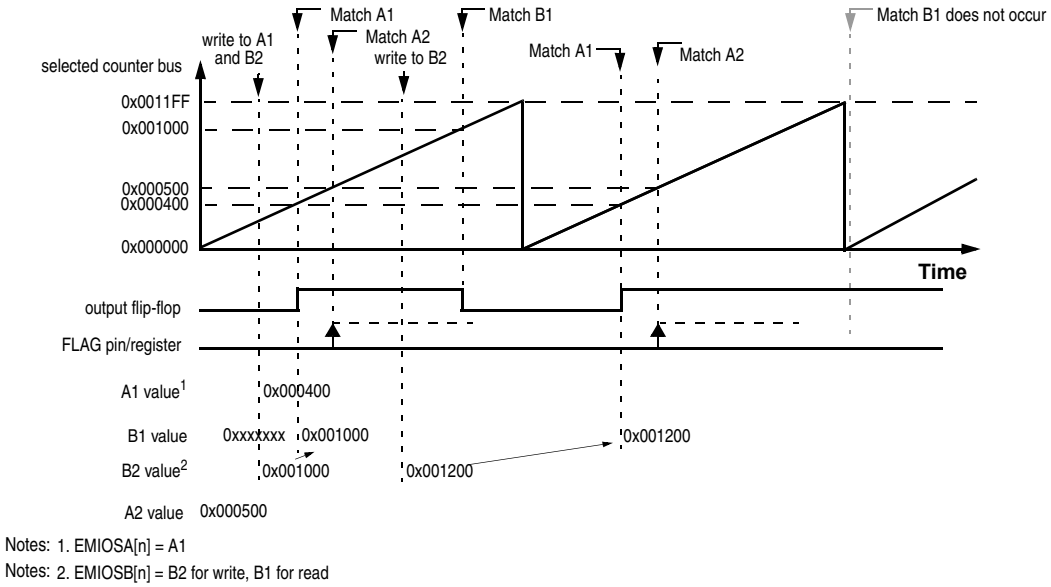


Figure 293. OPWMT with 100% duty cycle

Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the Unified Channel edge detector. A block diagram of the IPF is shown in *Figure 294*.

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[0:3] in EMIOSC[n] register.

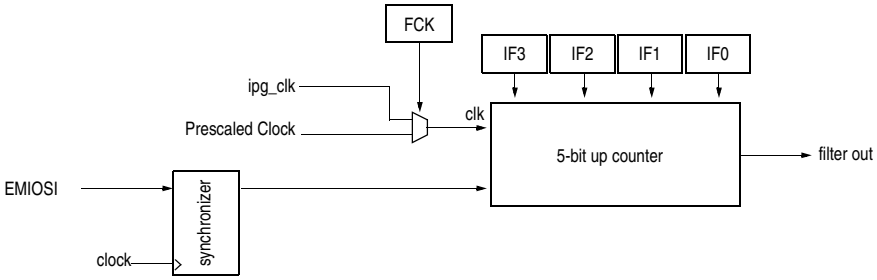


Figure 294. Input programmable filter submodule diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflows occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. A timing diagram of the input filter is shown in *Figure 295*.

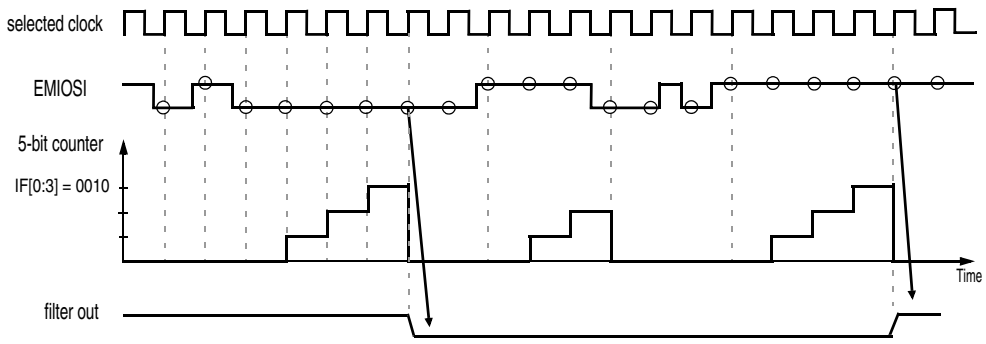


Figure 295. Input programmable filter example

The filter is not disabled during either freeze state or negated GTBE input.

Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the Unified Channels. The GCP output signal is prescaled by the value defined in *Figure 273* according to the UCPRE[0:1] bits in EMIOSC[n] register. The prescaler is enabled by setting the UCPREN bit in the EMIOSC[n] and can be stopped at any time by clearing this bit, thereby stopping the internal counter in the Unified Channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at both GPREN bit in EMIOSMCR register and UCPREN bit in EMIOSC[n] register, thus disabling prescalers;
2. Write the desired value for prescaling rate at UCPRE[0:1] bits in EMIOSC[n] register;
3. Enable channel prescaler by writing 1 at UCPREN bit in EMIOSC[n] register;
4. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

Effect of Freeze on the Unified Channel

When in debug mode, bit FRZ in the EMIOSMCR and bit FREN in the EMIOSC[n] register are both set, the internal counter and Unified Channel capture and compare functions are halted. The UC is frozen in its current state.

During freeze, all registers are accessible. When the Unified Channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

Note that for input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (FRZ in the EMIOSMCR or FREN in the EMIOSC[n] register) the channel actions resume, but may be inconsistent until channel enters GPIO mode again.

IP Bus Interface Unit (BIU)

The BIU provides the interface between the Internal Interface Bus (IIB) and the Peripheral Bus, allowing communication among all submodules and this IP interface.

The BIU allows 8, 16 and 32-bit access. They are performed over a 32-bit data bus in a single cycle clock.

Effect of Freeze on the BIU

When the FRZ bit in the EMIOSMCR is set and the module is in debug mode, the operation of BIU is not affected.

Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the channels. The main clock signal is prescaled by the value defined in *Figure 267* according to bits GPRE[0:7] in the EMIOSMCR. The global prescaler is enabled by setting the GPREN bit in the EMIOSMCR and can be stopped at any time by clearing this bit, thereby stopping the internal counters in all the channels.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write ‘0’ at GPREN bit in EMIOSMCR, thus disabling global prescaler;
2. Write the desired value for prescaling rate at GPRE[0:7] bits in EMIOSMCR;
3. Enable global prescaler by writing ‘1’ at GPREN bit in EMIOSMCR.

The prescaler is not disabled during either freeze state or negated GTBE input.

Effect of Freeze on the GCP

When the FRZ bit in the EMIOSMCR is set and the module is in debug mode, the operation of GCP submodule is not affected, that is, there is no freeze function in this submodule.

24.4.5 Initialization/Application information

On resetting the eMIOS the Unified Channels enter GPIO input mode.

Considerations

Before changing an operating mode, the UC must be programmed to GPIO mode and EMIOSA[n] and EMIOSB[n] registers must be updated with the correct values for the next operating mode. Then the EMIOSC[n] register can be written with the new operating mode. If a UC is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, that is, matches can occur in random time if the contents of EMIOSA[n] or EMIOSB[n] were not updated with the correct value before the time base matches the previous contents of EMIOSA[n] or EMIOSB[n].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

Application information

Correlated output signals can be generated by all output operation modes. Bits OU[n] of the EMIOSOUDIS register can be used to control the update of these output signals.

In order to guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio, but at a different clock cycle.

Time base generation

For MC with internal clock source operation modes, the internal counter rate can be modified by configuring the clock prescaler ratio. *Figure 296* shows an example of a time base with prescaler ratio equal to one.

Note: MCB and OPWFMB modes have a different behavior.

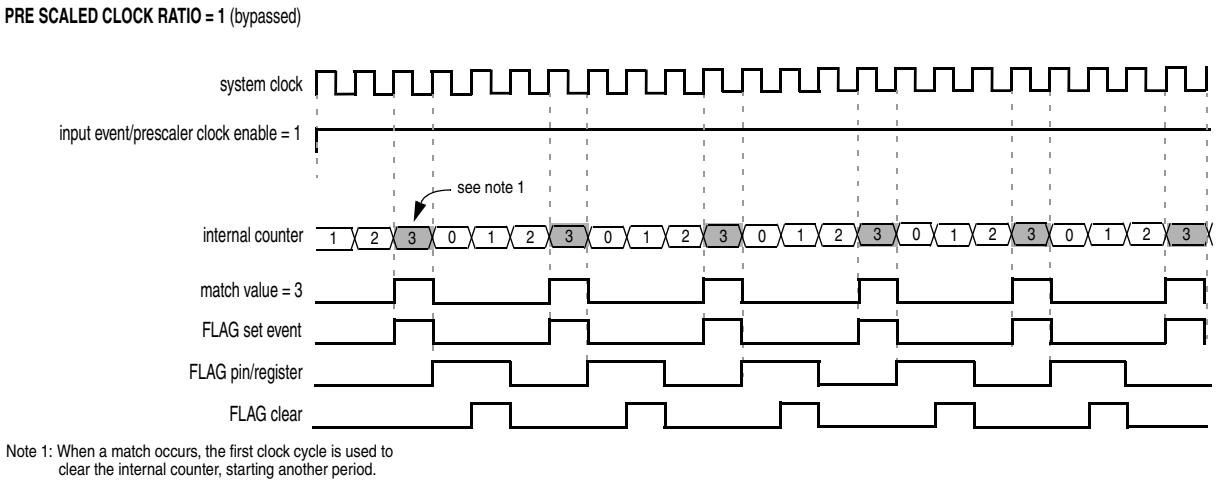
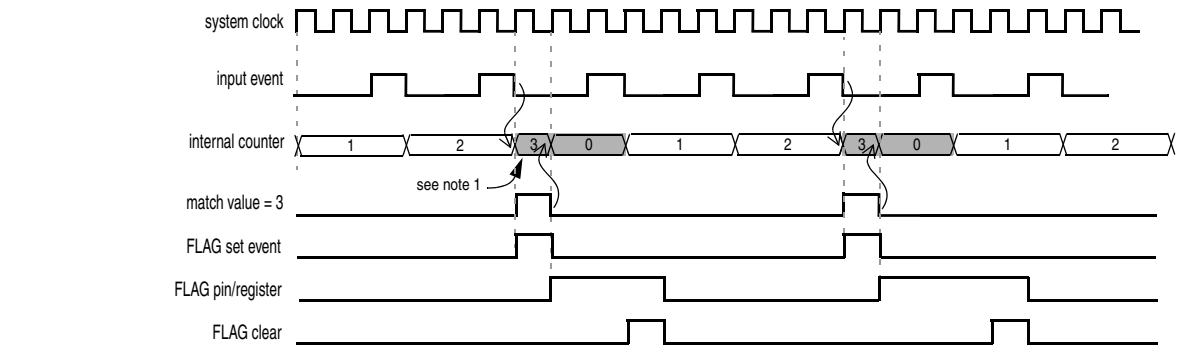


Figure 296. Time base period when running in the fastest prescaler ratio

If the prescaler ratio is greater than one or external clock is selected, the counter may behave in three different ways depending on the channel mode:

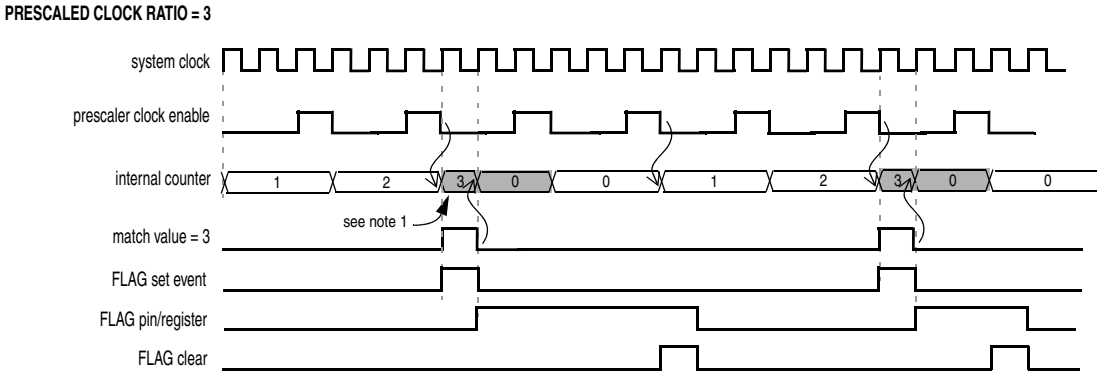
- *If MC mode and Clear on Match Start and External Clock source are selected the internal counter behaves as described in Figure 297.*
- *If MC mode and Clear on Match Start and Internal Clock source are selected the internal counter behaves as described in Figure 298.*
- *If MC mode and Clear on Match End are selected the internal counter behaves as described in Figure 299.*

Note: MCB and OPWFMB modes have a different behavior.



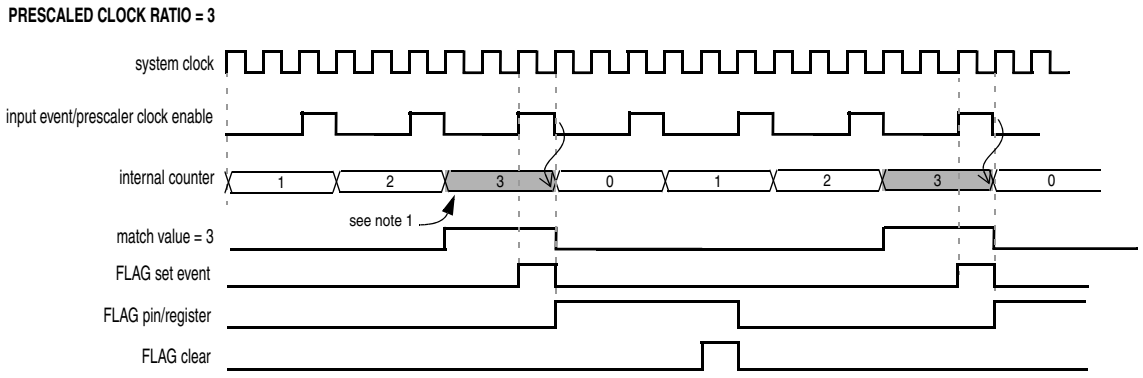
Note 1: When a match occurs, the first system clock cycle is used to clear the internal counter, and at the next edge of prescaler clock enable the counter will start counting.

Figure 297. Time base generation with external clock and clear on match start



Note 1: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of pre scaled clock the counter will start counting.

Figure 298. Time base generation with internal clock and clear on match start



Note 1: The match occurs only when the input event/prescaler clock enable is active.
Then, the internal counter is immediately cleared.

Figure 299. Time base generation with clear on match end

Coherent accesses

It is highly recommended that the software waits for a new FLAG set event before start reading EMIOSA[n] and EMIOSB[n] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt request or CTU trigger generation.

Reading the EMIOSA[n] register again in the same period of the last read of EMIOSB[n] register may lead to incoherent results. This will occur if the last read of EMIOSB[n] register occurred after a disabled B2 to B1 transfer.

Channel/Modes initialization

The following basic steps summarize basic output mode startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable Global Prescaler.
2. *[timebase channel]* Disable Channel Prescaler.
3. *[timebase channel]* Write initial value at internal counter.
4. *[timebase channel]* Set A/B register.
5. *[timebase channel]* Set channel to MC(B) Up mode.
6. *[timebase channel]* Set prescaler ratio.
7. *[timebase channel]* Enable Channel Prescaler.
8. *[output channel]* Disable Channel Prescaler.
9. *[output channel]* Set A/B register.
10. *[output channel]* Select timebase input through bits BSL[1:0].
11. *[output channel]* Enter output mode.
12. *[output channel]* Set prescaler ratio (same ratio as timebase channel).
13. *[output channel]* Enable Channel Prescaler.
14. *[global]* Enable Global Prescaler.
15. *[global]* Enable Global Time Base.

The timebase channel and the output channel may be the same for some applications such as in OPWFM(B) mode or whenever the output channel is intended to run the timebase itself.

The flags can be configured at any time.

24.5 Periodic Interrupt Timer (PIT)

24.5.1 Introduction

The PIT is an array of timers that can be used to raise interrupts.

Figure 300 shows the PIT block diagram.

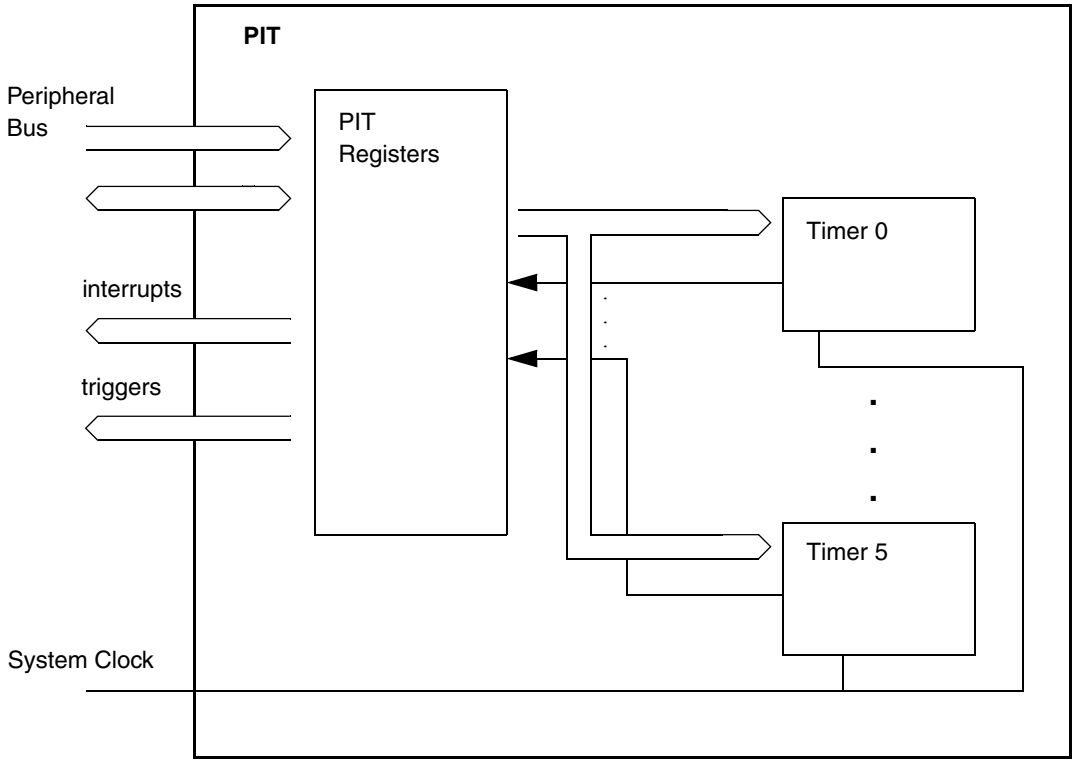


Figure 300. PIT block diagram

24.5.2 Features

The main features of this block are:

- *Timers can generate interrupts*
- *All interrupts are maskable*
- *Independent timeout periods for each timer*

24.5.3 Signal description

The PIT module has no external pins.

24.5.4 Memory map and register description

This section provides a detailed description of all registers accessible in the PIT module.

Memory map

Table 278 gives an overview of the PIT registers. See the chip memory map for the PIT base address.

Table 278. PIT memory map

Base address: 0xC3FF_0000		
Address offset	**Use**	**Location**
0x000	PIT Module Control Register (PITMCR)	*on page 24-572*
0x004–0x0FC	Reserved	
0x100–0x10C	Timer Channel 0	See *Table 279*
0x110–0x11C	Timer Channel 1	See *Table 279*
0x120–0x12C	Timer Channel 2	See *Table 279*
0x130–0x13C	Timer Channel 3	See *Table 279*
0x140–0x14C	Timer Channel 4	See *Table 279*
0x150–0x15C	Timer Channel 5	See *Table 279*

Table 279. Timer channel *n*

Address offset	Use	Location
channel + 0x00	Timer Load Value Register (LDVAL)	*on page 24-573*
channel + 0x04	Current Timer Value Register (CVAL)	*on page 24-574*
channel + 0x08	Timer Control Register (TCTRL)	*on page 24-574*
channel + 0x0C	Timer Flag Register (TFLG)	*on page 24-575*

Note: Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.

Note: Reserved registers will read as 0, writes will have no effect.

PIT Module Control Register (PITMCR)

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Figure 301. PIT Module Control Register (PITMCR)

Offset: 0x000 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	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 280. PITMCR field descriptions

Field	Description
MDIS	Module Disable This is 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 (LDVAL)

This register selects the timeout period for the timer interrupts.

Figure 302. Timer Load Value Register (LDVAL)

Offset: channel_base + 0x00 Access: Read/Write

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

Table 281. LDVAL field descriptions

Field	Description
TSV	Time Start Value This field sets the timer start value. The timer counts down until it reaches 0, then it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer, instead the value is 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 307*).

Current Timer Value Register (CVAL)

This register indicates the current timer position.

Figure 303. Current Timer Value Register (CVAL)

Offset: channel_base + 0x04 Access: Read

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

Table 282. CVAL field descriptions

Field	Description
TVL	Current Timer Value This field represents 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 246).*

Timer Control Register (TCTRL)

This register contains the control bits for each timer.

Figure 304. Timer Control Register (TCTRL)

Offset: channel_base + 0x08 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	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 283. TCTRL 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 (TFLG)

This register holds the PIT interrupt flags.

Figure 305. Timer Flag Register (TFLG)

Offset: channel_base + 0x0C 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	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 284. TFLG 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

24.5.5 Functional description

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 306*).

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 307*).

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 308*).

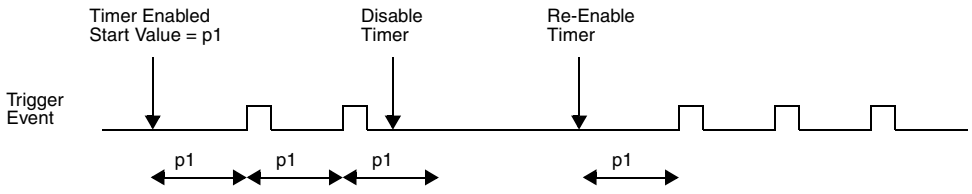


Figure 306. Stopping and starting a timer

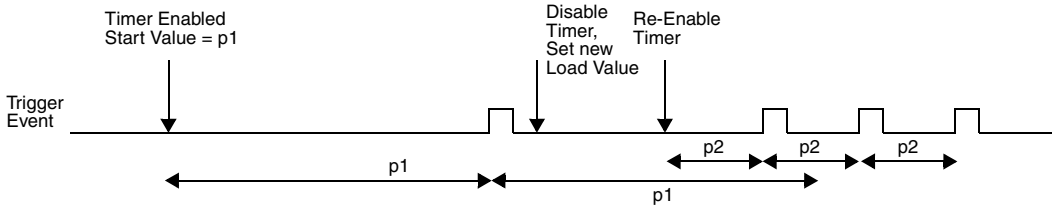


Figure 307. Modifying running timer period

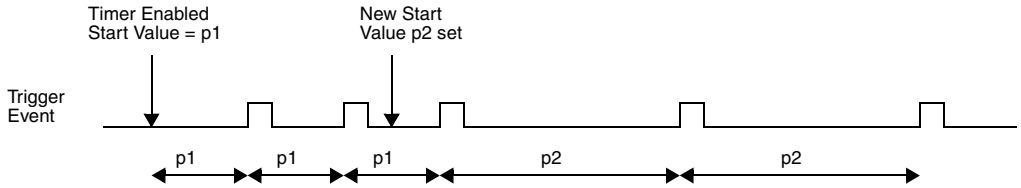


Figure 308. Dynamically setting a new load value

Debug mode

In Debug mode the timers will be frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (for example, the timer values) and then continue the operation.

Interrupts

All of the timers support interrupt generation. See the INTC chapter of the reference manual 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.

24.5.6 Initialization and application information

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 programming PIT_MCR[MDIS] = 0.

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 set as follows:

- *LDVAL for Timer 1 is set to 0x0003E7FF*
- *LDVAL for Timer 3 is set to 0x0016E35F*

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;


// 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 3 for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

25 Analog-to-Digital Converter (ADC)

25.1 Overview

25.1.1 Device-specific features

- 10-bit resolution
- 36 channels (depending on package type), expandable to 64 channels via external multiplexing
  - As many as 16 precision channels
  - As many as 20 standard channels, 4 being expandable to as many as 32 external channels
- Address decoder signal generation (alternate functions MA[2:0]) to control external multiplexers
- Individual conversion registers for each channel (internal and external)
- 3 different sampling and conversion time registers CTR[0:2] (internal precision channels, standard channels, external channels)
- As many as 64 data registers for storing converted data. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- Conversion triggering sources:
  - Software
  - CTU
  - PIT channel 2 (for injected conversion)
- 4 analog watchdogs
  - Interrupt capability
  - Allow continuous hardware monitoring of 4 analog input channels
- Presampling (V_{SS} and V_{DD})
- Conversions on external channels managed in the same way as internal channels, making it transparent to the application
- One Shot/Scan Modes
- Chain Injection Mode
- Power-down mode
- 2 different Abort functions allow to abort either single-channel conversion or chain conversion
- Auto-clock-off

25.1.2 Device-specific implementation

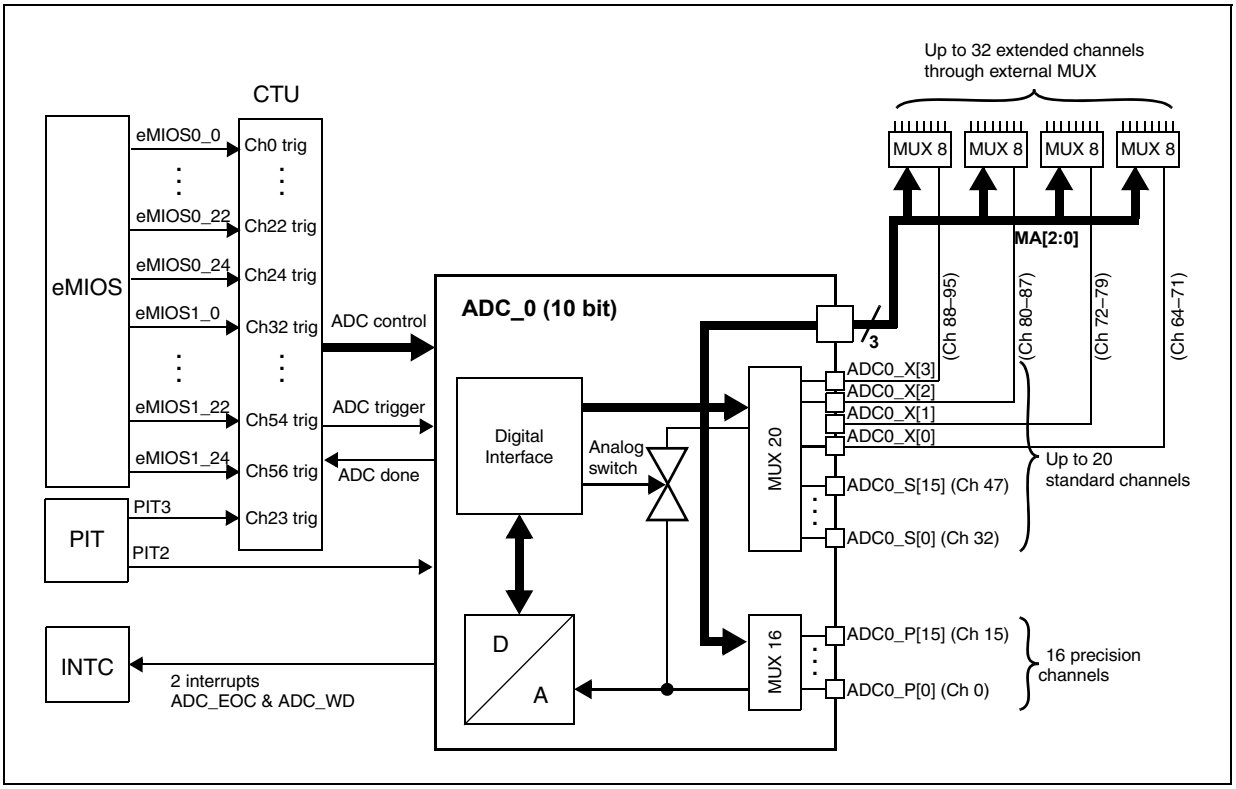


Figure 309. ADC implementation

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 (Cross Triggering Unit or PIT).

There are three types of input channels:

- Internal precision, ADC0_P[n] (internally multiplexed precision channels)
- Internal standard, ADC0_S[n] (internally multiplexed standard channels)
- External ADC0_X[n] (externally multiplexed standard channels)

The mask registers present within the ADC can be programmed to configure which channel has to be converted.

Three external decode signals MA[2:0] (multiplexer address) are provided for external channel selection and are available as alternate functions on GPIO.

The MA[0:2] are controlled by the ADC itself and are set automatically by the hardware.

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

Three conversion modes are available within the ADC:

- Normal conversion
- Injected conversion
- CTU triggered 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 (TRGEN reset)—If the external trigger enable bit is reset, the conversion chain starts when the MCR[NSTART] bit is set.
- By trigger (TRGEN set)—An on-chip internal signal triggers an ADC conversion. The settings in the MCR select how conversions are triggered based on these internal signals:
  - If the EDGLEV (edge/level selection) bit in the MCR is cleared, then a rising/falling edge (depending on the MCR[EDGE] bit) detected in the signal sets the MSR[NSTART] bit and starts the programmed conversion. EDGE = 0 selects a falling edge. EDGE = 1 selects a rising edge.
  - If the EDGLEV bit in the MCR is set, the conversion is started if and only if the MCR[NSTART] bit is set and the programmed level on the trigger signal is detected. The level is selected using the MCR[EDGE] bit. EDGE = 0 means that the start of conversion is enabled if the signal is low. If EDGE = 1, the start of conversion is enabled when the signal is high.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the

Table 285. Configurations for starting normal conversion

Type of conversion start	MCR				MSR	Result
	TRGEN	NSTART	EDGLEV	EDGE	NSTART	
Software	0	1	—	—	1	Conversion chain starts
Trigger	1	—	0	0	1	A falling edge detected in a trigger signal sets the NSTART bit in the MSR and starts the programmed conversion.
				1		A rising edge detected in a trigger signal sets the NSTART bit in the MSR and starts the programmed conversion.

Table 285. Configurations for starting normal conversion (continued)

Type of conversion start	MCR				MSR	Result
	TRGEN	NSTART	EDGLEV	EDGE	NSTART	
Trigger	1	1	1	0	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.
				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 high.

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 310*.

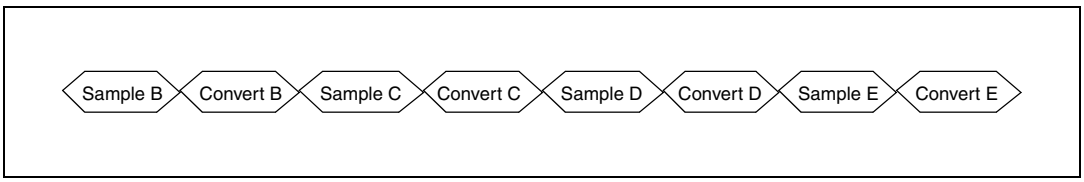


Figure 310. Normal conversion flow

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 3 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 4 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.

If the conversion is started by an external trigger and EDGLEV is '0', the MCR[NSTART] bit is not set. As a consequence, once started the only way to stop scan mode conversion is to set the MODE bit to '0'.

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 Normal conversion, each channel can be individually selected. This injected conversion (which can only occur 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 311*.

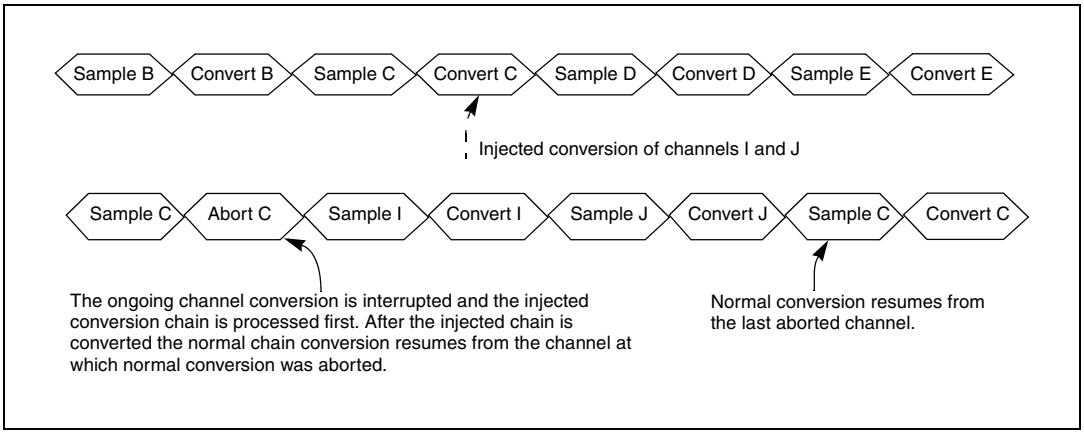


Figure 311. Injected sample/conversion sequence

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 PIT when MCR[JTRGEN] is set; a programmed event (rising/falling edge depending on MCR[JEDGE]) on the signal coming from PIT 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.

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 peripheral set 3 clock. Otherwise, the ADC clock is half of the peripheral set 3 clock 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.

25.3.3 ADC sampling and conversion timing

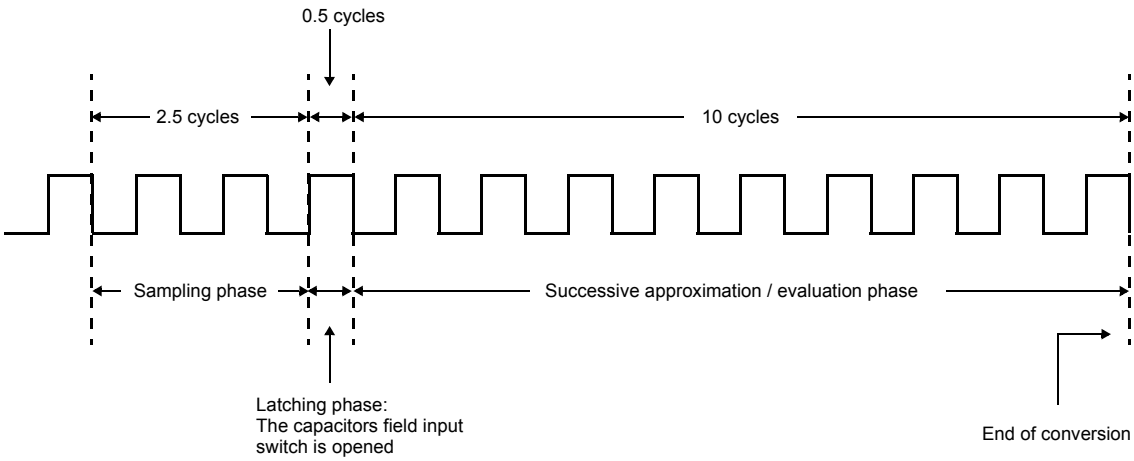
In order to support different loading and switching times, several different Conversion Timing registers (CTR) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

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 are used to 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 312 represents the sampling and conversion sequence.



Note: Operating conditions — INPLATCH = 0, INPSAMP = 3, INPCMP = 1 and Fadc clk = 20 MHz

Figure 312. 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 286. ADC sampling and conversion timing at 5 V / 3.3 V for ADC_0

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
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 287. Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC_0

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 is triggered by multiple input events (eMIOS and PIT) and can be used to select the channels to be converted from the appropriate event configuration register. A

single channel is converted for each request. After performing the conversion, the ADC returns the result on internal bus.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and the ADC are synchronous with the peripheral set 3 clock in both cases.

CTU in trigger mode

In CTU trigger mode, normal and injected conversions triggered by the CPU are still enabled.

Once the CTU event configuration register (CTU_EVTCFGRx) is configured and the corresponding trigger from the eMIOS or PIT is received, the conversion starts. The MSR[CTUSTART] is set automatically at this point and it is also automatically reset when the CTU triggered conversion is completed.

If an injected conversion (programmed by the user by setting the JSTART bit) is ongoing and CTU conversion is triggered, then the injected channel conversion chain is aborted and only the CTU triggered conversion proceeds. By aborting the injected conversion, the MSR[JSTART] is reset. That abort is signalled through the status bit MSR[JABORT].

If a normal conversion is ongoing and a CTU conversion is triggered, then any ongoing channel conversion is aborted and the CTU triggered conversion is processed. When it is finished, the normal conversion resumes from the channel at which the normal conversion was aborted.

If another CTU conversion is triggered before the end of the conversion, that request is discarded.

When a normal conversion is requested during CTU conversion (CTUSTART bit = ‘1’), the normal conversion starts when CTU conversion is completed (CTUSTART = ‘0’). Otherwise, when an Injected conversion is requested during CTU conversion, the injected conversion is discarded and the MCR[JSTART] is immediately reset.

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 313*, *Figure 314*).

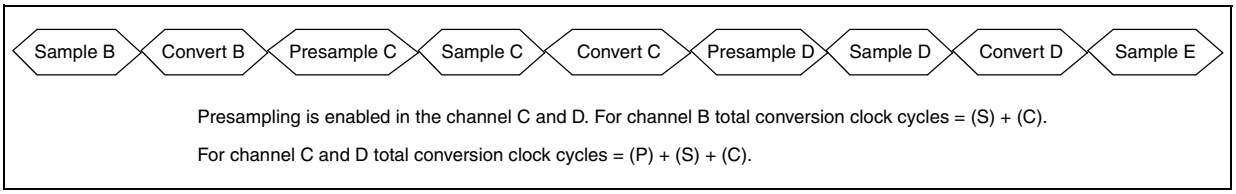


Figure 313. Presampling sequence

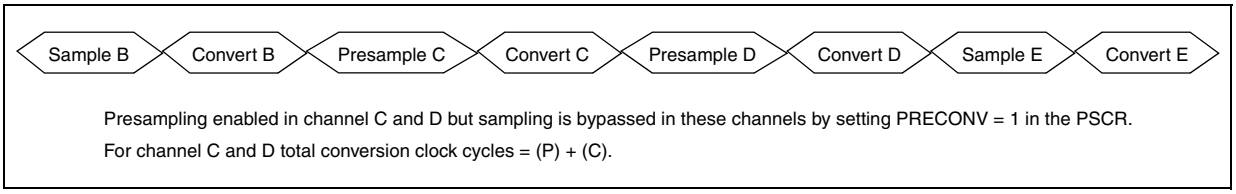


Figure 314. Presampling sequence with PRECONV = 1

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 288*.

Table 288. 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 315*) specified by an upper and a lower threshold value named THRH and THRL respectively.

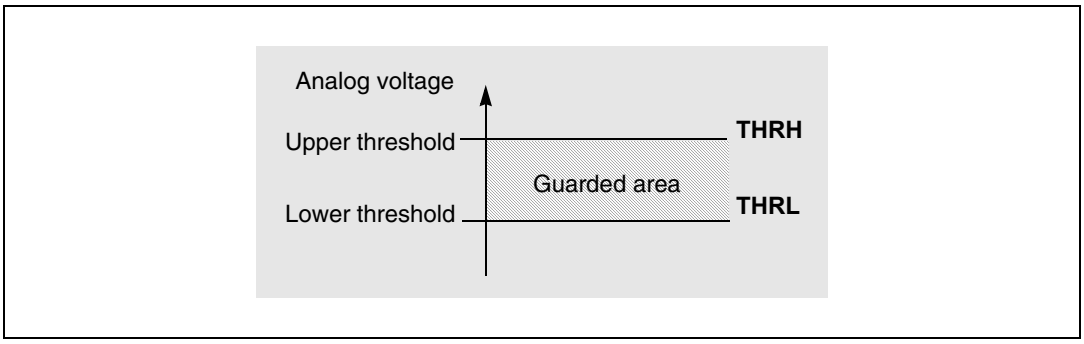


Figure 315. Guarded area

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 289*. Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

Table 289. 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 channel on which the analog watchdog is to be applied is selected by the TRC[THRCH]. 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 registers THRHLR.

For example, if channel number 3 is to be monitored with threshold values in THRHLR1, then the TRC[THRCH] 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] has to 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.

25.3.7 Interrupts

The ADC generates the following maskable interrupt signals:

- ADC_EOC interrupt requests
  - EOC (end of conversion)
  - ECH (end of chain)
  - JEOC (end of injected conversion)
  - JECH (end of injected chain)
  - EOCTU (end of CTU conversion)
- 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 CEOCFR[0..2]. Two registers named CEOCFR[0..2] (Channel Pending Registers) and IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt request to INT module.

Interrupts can be individually enabled on a channel by channel basis by programming the CIMR (Channel Interrupt Mask Register).

Several CEOCFR[0..2] are also provided in order to signal which of the channels' measurement has been completed.

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.

The CEOCFR[0..2] contains the interrupt pending request status. If the user wants to clear a particular interrupt event status, then writing a '1' to the corresponding status bit clears the pending interrupt flag (at this write operation all the other bits of the CEOCFR[0..2] must be maintained at '0').

25.3.8 External decode signals delay

The ADC provides several external decode signals to select which external channel has to be converted. In order to take into account the control switching time of the external analog multiplexer, a Decode Signals Delay register (DSDR) is provided. The delay between the decoding signal selection and the actual start of conversion can be programmed by writing the field DSD[0:7].

After having selected the channel to be converted, the MA[0:2] control lines are automatically reset. For instance, in the event of normal scan conversion on ANP[0] followed by ANX[0,7] (ADC ch 71) all the MA[0:2] bits are set and subsequently reset.

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 CSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set.

When CTU trigger mode is enabled, the application has to wait for the end of conversion (CTUSTART bit automatically reset).

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.

25.4 Register descriptions

25.4.1 Introduction

Table 290 lists ADC_0 registers with their address offsets and reset values.

Table 290. ADC_0 digital registers

Base address: 0xFFE0_0000		Location
Address offset	**Register name**	
0x0000	Main Configuration Register (MCR)	*on page 25-596*
0x0004	Main Status Register (MSR)	*on page 25-598*
0x0008 .. 0x000F	Reserved	—
0x0010	Interrupt Status Register (ISR)	*on page 25-600*
0x0014	Channel Pending Register (CEOCFR0)	*on page 25-600*
0x0018	Channel Pending Register (CEOCFR1)	*on page 25-600*
0x001C	Channel Pending Register (CEOCFR2)	*on page 25-600*

Table 290. ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0020	Interrupt Mask Register (IMR)	on page 25-602
0x0024	Channel Interrupt Mask Register (CIMR0)	on page 25-603
0x0028	Channel Interrupt Mask Register (CIMR1)	on page 25-603
0x002C	Channel Interrupt Mask Register (CIMR2)	on page 25-603
0x0030	Watchdog Threshold Interrupt Status Register (WTISR)	on page 25-605
0x0034	Watchdog Threshold Interrupt Mask Register (WTIMR)	on page 25-605
0x0038 .. 0x004F	Reserved	—
0x0050	Threshold Control Register 0 (TRC0)	on page 25-607
0x0054	Threshold Control Register 1 (TRC1)	on page 25-607
0x0058	Threshold Control Register 2 (TRC2)	on page 25-607
0x005C	Threshold Control Register 3 (TRC3)	on page 25-607
0x0060	Threshold Register 0 (THRHLR0)	on page 25-608
0x0064	Threshold Register 1 (THRHLR1)	on page 25-608
0x0068	Threshold Register 2 (THRHLR2)	on page 25-608
0x006C	Threshold Register 3 (THRHLR3)	on page 25-608
0x0080	Presampling Control Register (PSCR)	on page 25-608
0x0084	Presampling Register 0 (PSR0)	on page 25-609
0x0088	Presampling Register 1 (PSR1)	on page 25-609
0x008C	Presampling Register 2 (PSR2)	on page 25-609
0x0090 .. 0x0093	Reserved	—
0x0094	Conversion Timing Register 0 (CTR0)	on page 25-611

Table 290. ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0098	Conversion Timing Register 1 (CTR1)	on page 25-611
0x009C	Conversion Timing Register 2 (CTR2)	on page 25-611
0x00A0 .. 0x00A3	Reserved	—
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	on page 25-611
0x00A8	Normal Conversion Mask Register 1 (NCMR1)	on page 25-611
0x00AC	Normal Conversion Mask Register 2 (NCMR2)	on page 25-611
0x00B0 .. 0x00B3	Reserved	—
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	on page 25-614
0x00B8	Injected Conversion Mask Register 1 (JCMR1)	on page 25-614
0x00BC	Injected Conversion Mask Register 2 (JCMR2)	on page 25-614
0x00C0 .. 0x00C3	Reserved	—
0x00C4	Decode Signals Delay Register (DSDR)	on page 25-616
0x00C8	Power-down Exit Delay Register (PDEDR)	on page 25-616
0x00CC .. 0x00FF	Reserved	—
0x0100	Channel 0 Data Register (CDR0)	on page 25-617
0x0104	Channel 1 Data Register (CDR1)	on page 25-617
0x0108	Channel 2 Data Register (CDR2)	on page 25-617
0x010C	Channel 3 Data Register (CDR3)	on page 25-617
0x0110	Channel 4 Data Register (CDR4)	on page 25-617
0x0114	Channel 5 Data Register (CDR5)	on page 25-617
0x0118	Channel 6 Data Register (CDR6)	on page 25-617
0x011C	Channel 7 Data Register (CDR7)	on page 25-617

Table 290. ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0120	Channel 8 Data Register (CDR8)	on page 25-617
0x0124	Channel 9 Data Register (CDR9)	on page 25-617
0x0128	Channel 10 Data Register (CDR10)	on page 25-617
0x012C	Channel 11 Data Register (CDR11)	on page 25-617
0x0130	Channel 12 Data Register (CDR12)	on page 25-617
0x0134	Channel 13 Data Register (CDR13)	on page 25-617
0x0138	Channel 14 Data Register (CDR14)	on page 25-617
0x013C	Channel 15 Data Register (CDR15)	on page 25-617
0x0140 .. 0x017F	Reserved	—
0x0180	Channel 32 Data Register (CDR32)	on page 25-617
0x0184	Channel 33 Data Register (CDR33)	on page 25-617
0x0188	Channel 34 Data Register (CDR34)	on page 25-617
0x018C	Channel 35 Data Register (CDR35)	on page 25-617
0x0190	Channel 36 Data Register (CDR36)	on page 25-617
0x0194	Channel 37 Data Register (CDR37)	on page 25-617
0x0198	Channel 38 Data Register (CDR38)	on page 25-617
0x019C	Channel 39 Data Register (CDR39)	on page 25-617
0x01A0	Channel 40 Data Register (CDR40)	on page 25-617
0x01A4	Channel 41 Data Register (CDR41)	on page 25-617
0x01A8	Channel 42 Data Register (CDR42)	on page 25-617
0x01AC	Channel 43 Data Register (CDR43)	on page 25-617

Table 290. ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x01B0	Channel 44 Data Register (CDR44)	on page 25-617
0x01B4	Channel 45 Data Register (CDR45)	on page 25-617
0x01B8	Channel 46 Data Register (CDR46)	on page 25-617
0x01BC	Channel 47 Data Register (CDR47)	on page 25-617
0x01C0 .. 0x01FF	Reserved	—
0x0200	Channel 64 Data Register (CDR64)	on page 25-617
0x0204	Channel 65 Data Register (CDR65)	on page 25-617
0x0208	Channel 66 Data Register (CDR66)	on page 25-617
0x020C	Channel 67 Data Register (CDR67)	on page 25-617
0x0210	Channel 68 Data Register (CDR68)	on page 25-617
0x0214	Channel 69 Data Register (CDR69)	on page 25-617
0x0218	Channel 70 Data Register (CDR70)	on page 25-617
0x021C	Channel 71 Data Register (CDR71)	on page 25-617
0x0220	Channel 72 Data Register (CDR72)	on page 25-617
0x0224	Channel 73 Data Register (CDR73)	on page 25-617
0x0228	Channel 74 Data Register (CDR74)	on page 25-617
0x022C	Channel 75 Data Register (CDR75)	on page 25-617
0x0230	Channel 76 Data Register (CDR76)	on page 25-617
0x0234	Channel 77 Data Register (CDR77)	on page 25-617
0x0238	Channel 78 Data Register (CDR78)	on page 25-617
0x023C	Channel 79 Data Register (CDR79)	on page 25-617

Table 290. ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0240	Channel 80 Data Register (CDR80)	on page 25-617
0x0244	Channel 81 Data Register (CDR81)	on page 25-617
0x0248	Channel 82 Data Register (CDR82)	on page 25-617
0x024C	Channel 83 Data Register (CDR83)	on page 25-617
0x0250	Channel 84 Data Register (CDR84)	on page 25-617
0x0254	Channel 85 Data Register (CDR85)	on page 25-617
0x0258	Channel 86 Data Register (CDR86)	on page 25-617
0x025C	Channel 87 Data Register (CDR87)	on page 25-617
0x0260	Channel 88 Data Register (CDR88)	on page 25-617
0x0264	Channel 89 Data Register (CDR89)	on page 25-617
0x0268	Channel 90 Data Register (CDR90)	on page 25-617
0x026C	Channel 91 Data Register (CDR91)	on page 25-617
0x0270	Channel 92 Data Register (CDR92)	on page 25-617
0x0274	Channel 93 Data Register (CDR93)	on page 25-617
0x0278	Channel 94 Data Register (CDR94)	on page 25-617
0x027C	Channel 95 Data Register (CDR95)	on page 25-617
0x0280 .. 0x02FF	Reserved	—

25.4.2 Control logic registers

Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Figure 316. 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 291. 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 343* and *Figure 343*.
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 291. 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 317. 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 292. 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 001 Power-down 010 Wait state 011 Reserved 100 Sample 101 Reserved 110 Conversion 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 318. 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 293. 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.

Channel Pending Registers (CEOCFR[0..2])

CEOCFR0 = End of conversion pending interrupt for channel 0 to 15 (precision channels)

CEOCFR1 = End of conversion pending interrupt for channel 32 to 47 (standard channels)

CEOCFR2 = End of conversion pending interrupt for channel 64 to 95 (external multiplexed channels)

Figure 319. Channel Pending Register 0 (CEOCFR0)

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	EOC_CH15	EOC_CH14	EOC_CH13	EOC_CH12	EOC_CH11	EOC_CH10	EOC_CH9	EOC_CH8	EOC_CH7	EOC_CH6	EOC_CH5	EOC_CH4	EOC_CH3	EOC_CH2	EOC_CH1	EOC_CH0
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 320. Channel Pending Register 1 (CEOCFR1)

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	EOC_CH47	EOC_CH46	EOC_CH43	EOC_CH44	EOC_CH43	EOC_CH42	EOC_CH41	EOC_CH40	EOC_CH39	EOC_CH38	EOC_CH37	EOC_CH36	EOC_CH35	EOC_CH34	EOC_CH33	EOC_CH32
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 321. Channel Pending Register 2 (CEOCFR2)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EOC_CH95	EOC_CH94	EOC_CH93	EOC_CH92	EOC_CH91	EOC_CH90	EOC_CH89	EOC_CH88	EOC_CH87	EOC_CH86	EOC_CH85	EOC_CH84	EOC_CH83	EOC_CH82	EOC_CH81	EOC_CH80
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	EOC_CH79	EOC_CH78	EOC_CH77	EOC_CH76	EOC_CH75	EOC_CH74	EOC_CH73	EOC_CH72	EOC_CH71	EOC_CH70	EOC_CH69	EOC_CH68	EOC_CH67	EOC_CH66	EOC_CH65	EOC_CH64
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 294. CEOCFR field descriptions

Field	Description
EOC_CHn	When set, the measure of channel n is completed.

Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Figure 322. 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 295. Interrupt Mask Register (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.

Channel Interrupt Mask Register (CIMR[0..2])

CIMR0 = Enable bits for channel 0 to 15 (precision channels)

CIMR1 = Enable bits for channel 32 to 47 (standard channels)

CIMR2 = Enable bits for channel 64 to 95 (external multiplexed channels)

Figure 323. Channel Interrupt Mask Register 0 (CIMR0)

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

Figure 324. Channel Interrupt Mask Register 1 (CIMR1)

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	CIM 47	CIM 46	CIM 43	CIM 44	CIM 43	CIM 42	CIM 41	CIM 40	CIM 39	CIM 38	CIM 37	CIM 36	CIM 35	CIM 34	CIM 33	CIM 32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 325. Channel Interrupt Mask Register 2 (CIMR2)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIM 95	CIM 94	CIM 93	CIM 92	CIM 91	CIM 90	CIM 89	CIM 88	CIM 87	CIM 86	CIM 85	CIM 84	CIM 83	CIM 82	CIM 81	CIM 80
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	CIM 79	CIM 78	CIM 77	CIM 76	CIM 75	CIM 74	CIM 73	CIM 72	CIM 71	CIM 70	CIM 69	CIM 68	CIM 67	CIM 66	CIM 65	CIM 64
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 296. CIMR field descriptions

Field	Description
CIMn	Interrupt enable When set (CIMn = 1), interrupt for channel n is enabled.

Watchdog Threshold Interrupt Status Register (WTISR)

Figure 326. 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	WDG 3H	WDG 2H	WDG 1H	WDG 0H	WDG 3L	WDG 2L	WDG 1L	WDG 0L
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 297. 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 327. 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	MSK WDG 3H	MSK WDG 2H	MSK WDG 1H	MSK WDG 0H	MSK WDG 3L	MSK WDG 2L	MSK WDG 1L	MSK WDG 0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 298. 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 Threshold registers

Introduction

These four registers are used to store the user programmable lower and upper thresholds' values.

Threshold Control Register (TRCx, x = [0..3])

Figure 328. Threshold Control Register (TRCx, 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	0	0	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 299. TRCx field descriptions

Field	Description
THREN	Threshold enable When set, this bit enables the threshold detection feature for the selected channel.
THRCH	Choose the channel for threshold comparison.

Threshold Register (THRHLR[0:3])

The four THRHLR*n* registers are used to store the user-programmable thresholds' 10-bit values.

Figure 329. 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 300. THRHLRx field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

25.4.5 Presampling registers

Presampling Control Register (PSCR)

Figure 330. 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	PREVAL2		PREVAL1		PREVAL0		PRE CONV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 301. PSCR field descriptions

Field	Description
PREVAL2	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (external multiplexed channels). See *Table 288*.
PREVAL1	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (standard channels). See *Table 288*.
PREVAL0	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (precision channels). See *Table 288*.
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..2])

PSR0 = Enable bits of presampling for channel 0 to 15 (precision channels)

PSR1 = Enable bits of presampling for channel 32 to 47 (standard channels)

PSR2 = Enable bits of presampling for channel 64 to 95 (external multiplexed channels)

Figure 331. 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	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	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
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 332. Presampling Register 1 (PSR1)

Address: Base + 0x0088 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	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	47	46	43	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 333. Presampling Register 2 (PSR2)

Address: Base + 0x008C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
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	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 302. PSR field descriptions

Field	Description
PRESn	Presampling enable When set (PRESn = 1), presampling is enabled for channel n.

25.4.6 Conversion timing registers CTR[0..2]

CTR0 = associated to internal precision channels (from 0 to 15)

CTR1 = associated to standard channels (from 32 to 47)

CTR2 = associated to external multiplexed channels (from 64 to 95)

Figure 334. Conversion timing registers CTR[0..2]

Address: Base + 0x0094 (CTR0) / Base + 0x0098 (CTR1) / Base + 0x009C (CTR2) — 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 (1)		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1

1. Available only on CTR0

Table 303. 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 *Note: Available only on CTR0*
INPCMP	Configuration bits for comparison phase duration
INPSAMP	Configuration bits for sampling phase duration

25.4.7 Mask registers

Introduction

These registers are used to program which of the 96 input channels must be converted during Normal and Injected conversion.

Normal Conversion Mask Registers (NCMR[0..2])

NCMR0 = Enable bits of normal sampling for channel 0 to 15 (precision channels)

NCMR1 = Enable bits of normal sampling for channel 32 to 47 (standard channels)

NCMR2 = Enable bits of normal sampling for channel 64 to 95 (external multiplexed channels)

Figure 335. 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	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	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 336. Normal Conversion Mask Register 1 (NCMR1)

Address: Base + 0x00A8 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	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 337. Normal Conversion Mask Register 2 (NCMR2)

Address: Base + 0x00AC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
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	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 304. NCMR field descriptions

Field	Description
CHn	Sampling enable When set Sampling is enabled for channel n.

Note: The implicit channel conversion priority in the case in which all channels are selected is the following: ADC0_P[0:x], ADC0_S[0:y], ADC0_X[0:z].

The channels always start with 0, the lowest index.

Injected Conversion Mask Registers (JCMR[0..2])

JCMR0 = Enable bits of injected sampling for channel 0 to 15 (precision channels)

JCMR1 = Enable bits of injected sampling for channel 32 to 47(standard channels)

JCMR2 = Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels)

Figure 338. 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	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	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 339. Injected Conversion Mask Register 1 (JCMR1)

Address: Base + 0x00B8 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	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 340. Injected Conversion Mask Register 2 (JCMR2)

Address: Base + 0x00BC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
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	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 305. JCMR field descriptions

Field	Description
CHn	Sampling enable When set, sampling is enabled for channel n.

25.4.8 Delay registers

Decode Signals Delay Register (DSDR)

Figure 341. Decode Signals Delay Register (DSDR)

Address: Base + 0x00C4 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	DSD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 306. DSDR field descriptions

Field	Description
DSD	Delay between the external decode signals and the start of the sampling phase It is used to take into account the settling time of the external multiplexer. The decode signal delay is calculated as: DSD × 1/frequency of ADC clock. *Note: when ADC clock = Peripheral Clock/2 the DSD has to be incremented by 2 to see an additional ADC clock cycle delay on the decode signal.* *For example:* *DSD = 0; 0 ADC clock cycle delay* *DSD = 2; 1 ADC clock cycle delay* *DSD = 4; 2 ADC clock cycles delay*

Power-down Exit Delay Register (PDEDR)

Figure 342. 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 307. 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.

25.4.9 Data registers

Introduction

ADC conversion results are stored in data registers. There is one register per channel.

Channel Data Register (CDR[0..95])

CDR[0..15] = precision channels

CDR[32..47] = standard channels

CDR[64..95] = external multiplexed channels

Each data register also gives information regarding the corresponding result as described below.

Figure 343. Channel Data Register (CDR[0..95])

Address: See *Table 290* 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 308. 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.

Table 308. CDR field descriptions (continued)

Field	Description
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 0-95 converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this bitfield can be changed as shown in *Figure 343* and *Figure 343*.

26 Cross Triggering Unit (CTU)

26.1 Introduction

The Cross Triggering Unit (CTU) allows to synchronize an ADC conversion with a timer event from eMIOS (every mode which can generate a DMA request can trigger CTU) or PIT. To select which ADC channel must be converted on a particular timer event, the CTU provides the ADC with a 7-bit channel number. This channel number can be configured for each timer channel event by the application.

26.2 Main features

- Single cycle delayed trigger output. The trigger output is a combination of 64 (generic value) input flags/events connected to different timers in the system.
- One event configuration register dedicated to each timer event allows to define the corresponding ADC channel.
- Acknowledgment signal to eMIOS/PIT for clearing the flag
- Synchronization with ADC to avoid collision

26.3 Block diagram

The CTU block diagram is shown in *Figure 344*.

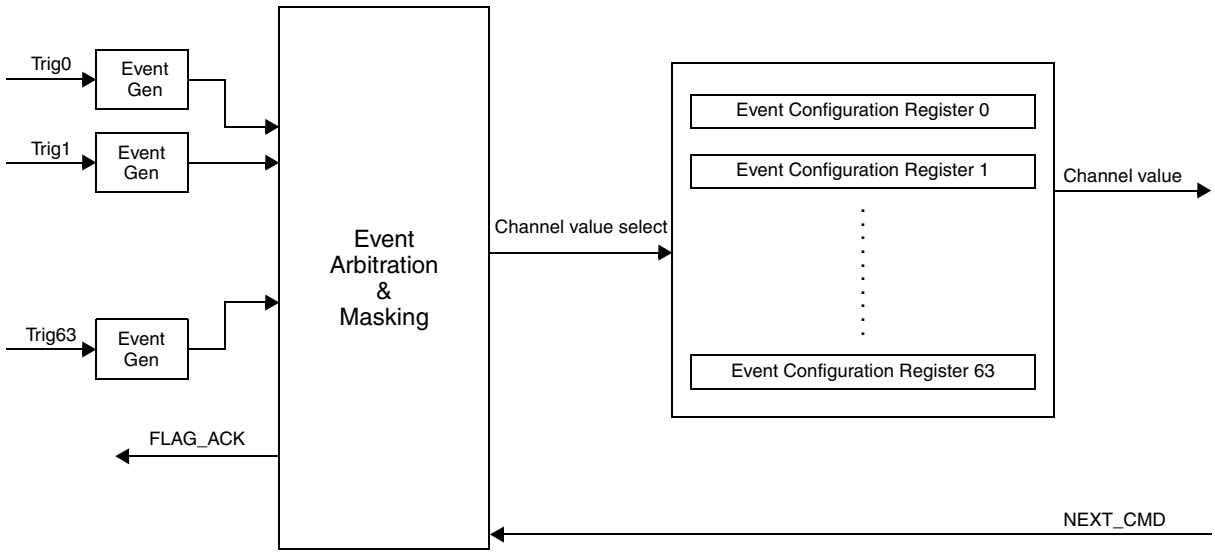


Figure 344. Cross Triggering Unit block diagram

26.4 Memory map and register descriptions

The CTU registers are listed in *Table 309*. Every register can have 32-bit access. The base address of the CTU is 0xFFE6_4000.

Table 309. CTU memory map

Base address: 0xFFE6_4000		
Address offset	**Register**	**Location**
0x000–0x02F	Reserved	
0x030–0x12C	*Event Configuration Registers 0..63* (CTU_EVTCFGR0..63)	*on page 26-620*

26.4.1 Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63)

Figure 345. Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63)

Offsets: 0x030–0x12C 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	TM	0	0	0	0	0	0	0	CLR_FLAG(1)	0	CHANNEL_VALUE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This bit implementation is generic based and implemented only for inputs mapped to PIT event flags.

Table 310. CTU_EVTCFGRx field descriptions

Field	Description
TM	Trigger Mask 0: Trigger masked 1: Trigger enabled
CLR_FLAG	To provide flag_ack through software 1: Flag_ack is forced to ‘1’ for the particular event 0: Flag_ack is dependent on flag servicing
CHANNEL_ VALUE	Channel value to be provided to ADC

These registers contain the ADC channel number to be converted when the timer event occurs. The CLR_FLAG is used to clear the respective timer event flag by software (this applies only to the PIT as the eMIOS flags are automatically cleared by the CTU).

The CLR_FLAG bit has to be used cautiously as setting this bit may result in a loss of events.

The event input can be masked by writing ‘0’ to bit TM of the CTU_EVTCFGR register. Writing ‘1’ to bit TM enables the CTU triggering for the corresponding eMIOS channel.

26.5 Functional description

This peripheral is used to synchronize ADC conversions with timer events (from eMIOS or PIT). When a timer event occurs, the CTU triggers an ADC conversion providing the ADC channel number to be converted. In case concurrent events occur the priority is managed according to the index of the timer event. The trigger output is a single cycle pulse used to trigger ADC conversion of the channel number provided by the CTU.

Each trigger input from the CTU is connected to the Event Trigger signal of an eMIOS channel. The assignment between eMIOS outputs and CTU trigger inputs is defined in *Table 311*.

Table 311. Trigger source

CTU trigger No.	Module	Source
0	eMIOS 0	Channel_0
1	eMIOS 0	Channel_1
2	eMIOS 0	Channel_2
3	eMIOS 0	Channel_3
4	eMIOS 0	Channel_4
5	eMIOS 0	Channel_5
6	eMIOS 0	Channel_6
7	eMIOS 0	Channel_7
8	eMIOS 0	Channel_8
9	eMIOS 0	Channel_9
10	eMIOS 0	Channel_10
11	eMIOS 0	Channel_11
12	eMIOS 0	Channel_12
13	eMIOS 0	Channel_13
14	eMIOS 0	Channel_14
15	eMIOS 0	Channel_15
16	eMIOS 0	Channel_16
17	eMIOS 0	Channel_17
18	eMIOS 0	Channel_18
19	eMIOS 0	Channel_19

Table 311. Trigger source (continued)

CTU trigger No.	Module	Source
20	eMIOS 0	Channel_20
21	eMIOS 0	Channel_21
22	eMIOS 0	Channel_22
23	PIT	PIT_3
24	eMIOS 0	Channel_24
25	Reserved	
26	Reserved	
27	Reserved	
28	Reserved	
29	Reserved	
30	Reserved	
31	Reserved	
32	eMIOS 1	Channel_0
33	eMIOS 1	Channel_1
34	eMIOS 1	Channel_2
35	eMIOS 1	Channel_3
36	eMIOS 1	Channel_4
37	eMIOS 1	Channel_5
38	eMIOS 1	Channel_6
39	eMIOS 1	Channel_7
40	eMIOS 1	Channel_8
41	eMIOS 1	Channel_9
42	eMIOS 1	Channel_10
43	eMIOS 1	Channel_11
44	eMIOS 1	Channel_12
45	eMIOS 1	Channel_13
46	eMIOS 1	Channel_14
47	eMIOS 1	Channel_15
48	eMIOS 1	Channel_16
49	eMIOS 1	Channel_17
50	eMIOS 1	Channel_18
51	eMIOS 1	Channel_19
52	eMIOS 1	Channel_20
53	eMIOS 1	Channel_21
54	eMIOS 1	Channel_22

Table 311. Trigger source (continued)

CTU trigger No.	Module	Source
55	Reserved	
56	eMIOS 1	Channel_24

Each event has a dedicated configuration register (CTU_EVTCFGR). These registers store a channel number which is used to communicate which channel needs to be converted.

In case several events are pending for ADC request, the priority is managed according to the timer event index. The lowest index has the highest priority. Once an event has been serviced (conversion requested to ADC) the eMIOS flag is cleared by the CTU and next prior event is handled.

The acknowledgment signal can be forced to '1' by setting the CLR_FLAG bit of the CTU_EVTCFGR register. These bits are implemented for only those input flags to which PIT flags are connected. Providing these bits offers the option of clearing PIT flags by software.

26.5.1 Channel value

The channel value stored in an event configuration register is demultiplexed to 7 bits and then provided to the ADC.

The mapping of the channel number value to the corresponding ADC channel is provided in *Table 311*.

Table 312. CTU-to-ADC channel assignment

10-bit ADC signal name	10-bit ADC channel #	Channel number in CTU_EVTCFGRx
ADC_P[0]	CH0	0
ADC_P[1]	CH1	1
ADC_P[2]	CH2	2
ADC_P[3]	CH3	3
ADC_P[4]	CH4	4
ADC_P[5]	CH5	5
ADC_P[6]	CH6	6
ADC_P[7]	CH7	7
ADC_P[8]	CH8	8
ADC_P[9]	CH9	9
ADC_P[10]	CH10	10
ADC_P[11]	CH11	11
ADC_P[12]	CH12	12
ADC_P[13]	CH13	13
ADC_P[14]	CH14	14
ADC_P[15]	CH15	15

Table 312. CTU-to-ADC channel assignment (continued)

10-bit ADC signal name	10-bit ADC channel #	Channel number in CTU_EVTCFGRx
ADC_S[0]	CH32	16
ADC_S[1]	CH33	17
ADC_S[2]	CH34	18
ADC_S[3]	CH35	19
ADC_S[4]	CH36	20
ADC_S[5]	CH37	21
ADC_S[6]	CH38	22
ADC_S[7]	CH39	23
ADC_S[8]	CH40	24
ADC_S[9]	CH41	25
ADC_S[10]	CH42	26
ADC_S[11]	CH43	27
ADC_S[12]	CH44	28
ADC_S[13]	CH45	29
ADC_S[14]	CH46	30
ADC_S[15]	CH47	31
ADC_X[0]	CH64 : CH71	32 : 39
ADC_X[1]	CH72 : CH79	40 : 47
ADC_X[2]	CH80 : CH87	48 : 55
ADC_X[3]	CH88 : CH95	56 : 63

CTU channel mapping should be taken into consideration when programming an event configuration register. For example, if the channel value of any event configuration register is programmed to 16, it will actually correspond to ADC channel 32 and conversion will occur for this channel.

27 Static RAM (SRAM)

27.1 Introduction

The general-purpose SRAM has a size of 48 KB. In every mode other than STANDBY all the 48 KB of SRAM are powered, while during STANDBY mode the user can decide to retain 32 KB or just 8 KB. See the MC_ME chapter in this reference manual for details.

The SRAM provides the following features:

- SRAM can be read/written from any bus master
- Byte, halfword and word addressable
- ECC (error correction code) protected with single-bit correction and double-bit detection

27.2 Low power configuration

In order to reduce leakage a portion of the SRAM can be switched off/unpowered during standby mode.

Table 313. Low power configuration

Mode	Configuration
RUN, TEST, SAFE and STOP	The entire SRAM is powered and operational.
STANDBY	Either 32 KB or just 8 KB of the SRAM remains powered. This option is software-selectable.

27.3 Register memory map

The L2SRAM occupies 48 KB of memory starting at the base address as shown in *Table 314*.

Table 314. SRAM memory map

Address	Register name	Register description	Size
0x4000_0000 (Base)	—	SRA	up to 48 KB

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM (see the *Error Correction Status Module (ECSM)* chapter of the reference manual for more information).

27.4 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.

27.4.1 Access timing

The system bus is a two-stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock cycle. *Table 315* 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 315. 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
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	

27.4.2 Reset effects on SRAM accesses

Asynchronous reset will possibly corrupt SRAM 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. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed. In case of no access ongoing when reset occurs, the SRAM corruption does not happen.

Instead, synchronous reset (SW reset) should be used in controlled function (without SRAM accesses) in case an initialization procedure without SRAM initialization is needed.

27.5 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 to any read accesses. This is also true for implicit read accesses caused by any write accesses of less than 32 bits as discussed in *Section 27.4 SRAM ECC mechanism*.

27.6 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. See *Section 27.4 SRAM ECC mechanism*.

28 Flash Memory

28.1 Introduction

The flash memory comprises a platform flash memory controller (PFlash) interface and the following flash memory arrays:

- One array of 512 KB for code (CFlash)
- One array of 64 KB for data (DFlash)

The flash memory architecture of this device is illustrated in *Figure 346*.

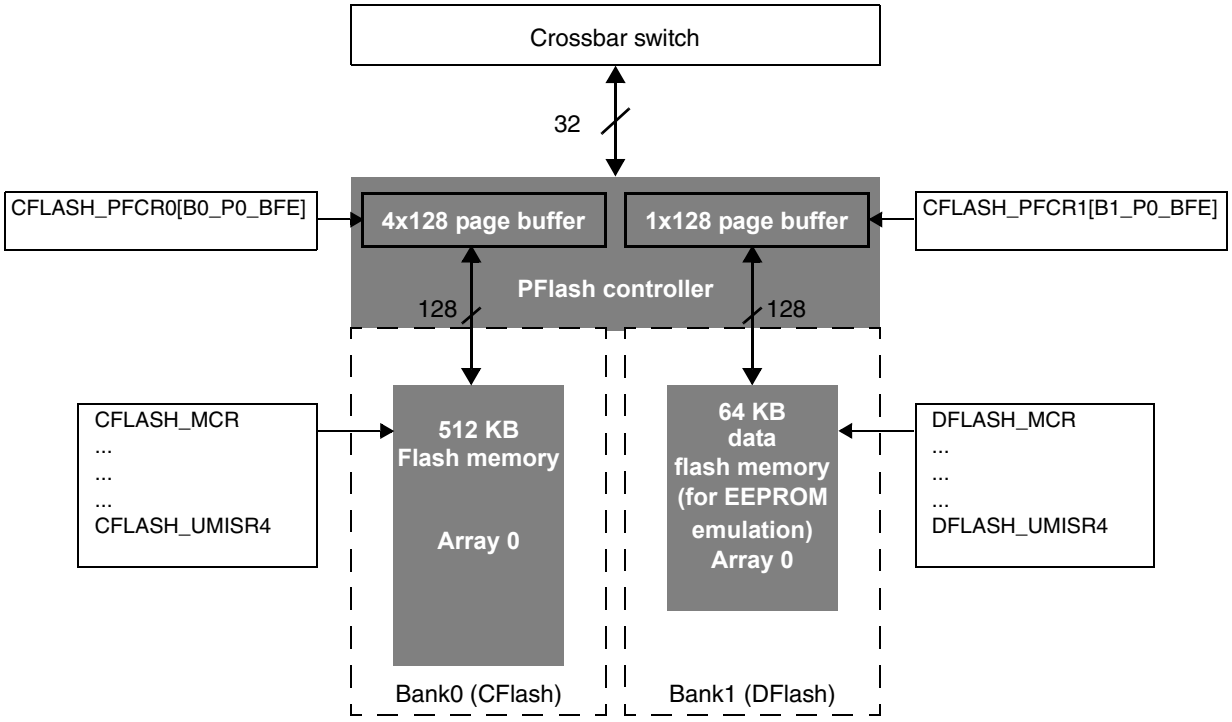


Figure 346. Flash memory architecture

The primary function of the flash memory module is to serve as electrically programmable and erasable nonvolatile memory.

Nonvolatile memory may be used for instruction and/or data storage.

The module is a nonvolatile solid-state silicon memory device consisting of:

- Blocks (also called “sectors”) of single transistor storage elements
- An electrical means for selectively adding (programming) and removing (erasing) charge from these elements
- A means of selectively sensing (reading) the charge stored in these elements

The flash memory module is arranged as two functional units:

- The flash memory core
- The memory interface

The flash memory core is composed of arrayed nonvolatile storage elements, sense amplifiers, row decoders, column decoders and charge pumps. The arrayed storage elements in the flash memory core are subdivided 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 memory core. The memory interface is also the interface between the flash memory module and a platform flash memory controller. It contains the ECC logic and redundancy logic.

A platform flash memory controller connects the flash memory module to a system bus, and contains all system level customization required for the device application.

28.2 Main features

Table 316. Flash memory features

Feature	CFlash	DFlash
High read parallelism (128 bits)	Yes	
Error Correction Code (SEC-DED) to enhance data retention	Yes	
Double Word Program (64 bits)	Yes	
Sector erase	Yes	
Single bank—Read-While-Write (RWW)	No	
Erase Suspend	Yes	
Program Suspend	No	
Software programmable program/erase protection to avoid unwanted writings	Yes	
Censored Mode against piracy	Yes	
Shadow Sector available	Yes	No
One-Time Programmable (OTP) area in Test Flash block	Yes	
Boot sectors	Yes	No

28.3 Block diagram

The flash memory module contains one Matrix Module, composed of a single bank (Bank 0) normally used for code storage. RWW operations are not possible.

Modify operations are managed by an embedded Flash Memory Program/Erase Controller (FPEC). Commands to the FPEC are given through a User Registers Interface.

The read data bus is 128 bits wide, while the flash memory registers are on a separate bus 32 bits wide addressed in the user memory map.

The high voltages needed for program/erase operations are generated internally.

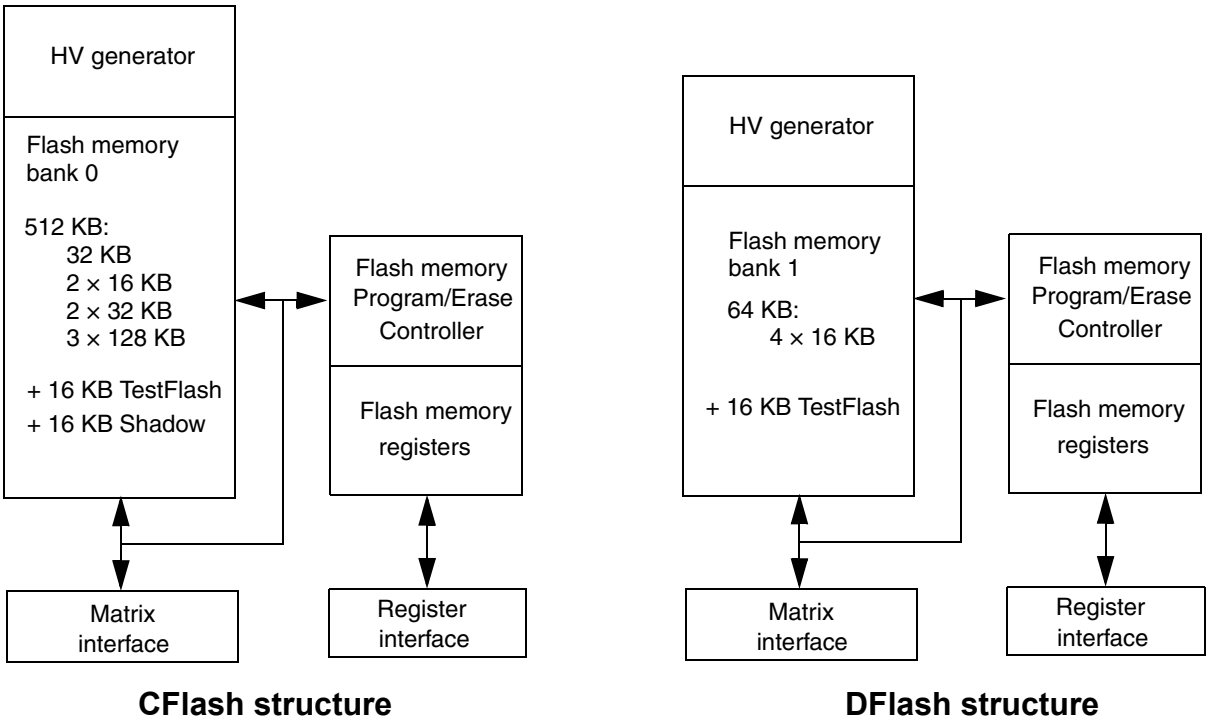


Figure 347. CFlash and DFlash module structures

28.4 Functional description

28.4.1 Module structure

The flash memory module is addressable by Double Word (64 bits) for program, and page (128 bits) for read. Reads to the flash memory always return 128 bits, although read page buffering may be done in the platform flash memory controller.

Each read of the flash memory module retrieves a page, or four consecutive words (128 bits) of information. The address for each word retrieved within a page differs from the other addresses in the page only by address bits (3:2).

The flash memory module supports fault tolerance through Error Correction Code (ECC) or error detection, or both. The ECC implemented within the flash memory module will correct single bit failures and detect double bit failures.

The flash memory module uses an embedded hardware algorithm implemented in the Memory Interface to program and erase the flash memory core.

The embedded hardware algorithm includes control logic that works with software block enables and software lock mechanisms to guard against accidental program/erase.

The hardware algorithm performs the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to guarantee data integrity and reliability.

In the flash memory module, logic levels are defined as follows:

- A programmed bit reads as logic level 0 (or low).
- An erased bit reads as logic level 1 (or high).

Program and erase of the flash memory module requires multiple system clock cycles to complete.

The erase sequence may be suspended.

The program and erase sequences may be aborted.

28.4.2 Flash memory module sectorization

CFlash module sectorization

The CFlash module supports 512 KBof 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 and censorship settings.

The module is composed of a single bank (Bank 0): Read-While-Write is not supported.

Bank 0 of the module is divided in 10 sectors including a reserved sector, named TestFlash, in which some One-Time Programmable (OTP) user data are stored, as well as a Shadow Sector in which user erasable configuration values can be stored.

The matrix module sectorization is shown in *Table 317*.

Table 317. CFlash module sectorization

Bank	Sector	Addresses	Size (KB)	Address space	CFLASH_LML field for locking the address space
0	0	0x00000000–0x00007FFF	32	Low	LLK0
	1	0x00008000–0x0000BFFF	16		LLK1
	2	0x0000C000–0x0000FFFF	16		LLK2
	3	0x00010000–0x00017FFF	32		LLK3
	4	0x00018000–0x0001FFFF	32		LLK4
	5	0x00020000–0x0003FFFF	128		LLK5
	6	0x00040000–0x0005FFFF	128	Mid	MLK0
	7	0x00060000–0x0007FFFF	128		MLK1
	Shadow	0x00200000–0x00203FFF	16	Shadow	TSLK
	Test	0x00400000–0x00403FFF	16	Test	TSLK

The division into blocks of the flash memory module is also used to implement independent erase/program protection. A software mechanism is provided to independently lock/unlock each block in low and mid address space against program and erase.

DFlash module sectorization

The DFlash module supports 64 KB of user memory, plus 16 KB of test memory (a portion of which is One-Time Programmable by the user).

The module is composed of a single bank (Bank 0): Read-While-Write is not supported.

Bank 0 of the 80 KB module is divided in four sectors. Bank 0 also contains a reserved sector named TestFlash in which some One-Time Programmable user data are stored.

The sectorization of the 80 KB matrix module is shown in *Table 318*.

Table 318. DFlash module sectorization

Bank	Sector	Addresses	Size (KB)	Address space	DFLASH_LML field for locking the address space
0	0	0x00800000–0x00803FFF	16	Low	LLK0
	1	0x00804000–0x00807FFF			LLK1
	2	0x00808000–0x0080BFFF			LLK2
	3	0x0080C000–0x0080FFFF			LLK3
	Test	0x00C00000–0x00C03FFF		Test	TSLK

The flash memory module is divided into blocks also to implement independent erase/program protection. A software mechanism is provided to independently lock/unlock each block in low and mid address space against program and erase.

28.4.3 TestFlash block

A TestFlash block is available in both the CFlash and DFlash modules. 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 to also support systems which require nonvolatile memory for security or to store system initialization information, or both.

A section of the TestFlash is reserved to store the nonvolatile information related to Redundancy, Configuration and Protection.

The ECC is also applied to TestFlash.

The structure of the TestFlash sector is detailed in *Table 319* and *Table 320*.

Table 319. CFlash TestFlash structure

Name	Description	Addresses	Size
—	User OTP area	0x400000–0x401FFF	8192 bytes
—	Reserved	0x402000–0x403CFF	7424 bytes
—	User OTP area	0x403D00–0x403DE7	232 bytes
CFLASH_NVLML	CFlash Nonvolatile Low/Mid Address Space Block Locking Register	0x403DE8–0x403DEF	8 bytes
—	Reserved	0x403DF0–0x403DF7	8 bytes
CFLASH_NVSLL	CFlash Nonvolatile Secondary Low/mid Address Space Block Locking Register	0x403DF8–0x403DFF	8 bytes
—	User OTP area	0x403E00–0x403EFF	256 bytes
—	Reserved	0x403F00–0x403FFF	256 bytes

Table 320. DFlash TestFlash structure

Name	Description	Addresses	Size
—	User OTP area	0xC00000–0xC01FFF	8192 bytes
—	Reserved	0xC02000–0xC03CFF	7424 bytes
—	User OTP area	0xC03D00–0xC03DE7	232 bytes
DFLASH_NVLML	DFlash Nonvolatile Low/Mid Address Space Block Locking Register	0xC03DE8–0xC03DEF	8 bytes
—	Reserved	0xC03DF0–0xC03DF7	8 bytes
DFLASH_NVSLL	DFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register	0xC03DF8–0xC03DFF	8 bytes
—	User OTP area	0xC03E00–0xC03EFF	256 bytes
—	Reserved	0xC03F00–0xC03FFF	256 bytes

Erase of the TestFlash block is always locked.

User mode program of the TestFlash block are enabled only when MCR[PEAS] is high.

The TestFlash block may be locked/unlocked against program by using the LML[TSLK] and SLL[STSLK] registers.

Programming of the TestFlash block has similar restrictions as the array in terms of how ECC is calculated. Only one programming operation is allowed per 64-bit ECC segment.

The first 8 KB of TestFlash block may be used for user defined functions (possibly to store serial numbers, other configuration words or factory process codes). Locations of the TestFlash other than the first 8 KB of OTP area cannot be programmed by the user application.

28.4.4 Shadow sector

The shadow sector is only present in the CFlash module.

User Mode program and erase of the shadow sector are enabled only when CFLASH_MCR[PEAS] is high.

The shadow sector may be locked/unlocked against program or erase by using the CFLASH_LML[TSLK] and CFLASH_SLL[STSLK] fields.

Programming of the shadow sector has similar restrictions as the array in terms of how ECC is calculated. Only one programming operation is allowed per 64-bit ECC segment between erases.

Erase of the shadow sector is done similarly to a sector erase.

The shadow sector contains specified data that are needed for user features.

The user area of shadow sector may be used for user defined functions (possibly to store boot code, other configuration words or factory process codes).

The structure of the shadow sector is detailed in *Table 321*.

Table 321. Shadow sector structure

Name	Description	Addresses	Size (bytes)
—	User area	0x200000–0x203DCF	15824
—	Reserved	0x203DD0–0x203DD7	8
NVPWD0–1	Nonvolatile Private Censorship PassWord 0–1 registers	0x203DD8–0x203DDF	8
NVSCC0–1	Nonvolatile System Censorship Control 0–1 registers	0x203DE0–0x203DE7	8
—	Reserved	0x203DE8–0x203DFF	24
NVPFAPR	Nonvolatile Platform Flash Memory Access Protection Register	0x203E00–0x203E07	8
—	Reserved	0x203E08–0x203E17	16
NVUSRO	Nonvolatile User Options register	0x203E18–0x203E1F	8
—	Reserved	0x203E20–0x203FFF	480

28.4.5 User mode operation

In User Mode the flash memory module may be read and written (register writes and interlock writes), programmed or erased.

The default state of the flash memory module is read.

The main, shadow and test address space can be read only in the read state.

The majority of CFlash and DFlash memory-mapped registers can be read even when the CFlash or DFlash is in power-down or low-power mode. The exceptions are as follows:

- CFlash
  - UT0[MRE, MRV, AIS, DSI0:7]
  - UT1
  - UT2
- DFlash
  - UT0[MRE, MRV, AIS, DSI0:7]
  - UT1
  - UT2

The flash memory 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 the ERS and ESUS fields in the corresponding MCR (CFLASH_MCR or DFLASH_MCR) are 1 and the PGM field is 0 (Erase Suspend).

Flash memory core reads return 128 bits (1 Page = 2 Double Words).

Registers reads return 32 bits (1 Word).

Flash memory core reads are done through the platform flash memory controller.

Registers reads to unmapped register address space will return all 0's.

Registers writes to unmapped register address space will have no effect.

Attempted array reads to invalid locations will result in indeterminate data. Invalid locations occur when blocks that do not exist in non 2^n array sizes are addressed.

Attempted interlock writes 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 cycle on the Flash Matrix and Read/Write cycles on the registers are possible. On the contrary, registers read/write accesses simultaneous to a Flash Matrix interlock write are forbidden.

28.4.6 Reset

A reset is the highest priority operation for the flash memory module and terminates all other operations.

The flash memory module uses reset to initialize register and status bits to their default reset values. If the flash memory module is executing a Program or Erase operation (PGM = 1 or ERS = 1 in CFLASH_MCR or DFLASH_MCR) and a reset is issued, the operation will be suddenly terminated and the module will disable the high voltage logic without damage to the high voltage circuits. Reset terminates all operations and forces the flash memory module into User Mode ready to receive accesses. Reset and power-off must not be used as a systematic way to terminate a Program or Erase operation.

After reset is negated, 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 the DONE field (in CFLASH_MCR or DFLASH_MCR) transitions. The DONE field may be polled to determine if the flash memory module has transitioned out of reset. Notice that the registers cannot be written until the DONE field is high.

28.4.7 Power-down mode

All flash memory DC current sources can be turned off in power-down mode, so that all power dissipation is due only to leakage in this mode. Flash memory power-down mode can be selected at ME_<mode>_MC.

Reads from or writes to the module are not possible in power-down mode.

When enabled the flash memory module returns to its pre-disable state in all cases unless in the process of executing an erase high voltage operation at the time of disable.

If the flash memory module is disabled during an erase operation, MCR[ESUS] bit is programmed to ‘1’. The user may resume the erase operation at the time the module is enabled by programming MCR[ESUS] = 0. MCR[EHV] must be high to resume the erase operation.

If the flash memory module is disabled during a program operation, the operation will in any case be completed and the power-down mode will be entered only after the programming ends.

The user should realize that, if the flash memory module is put in power-down mode and the interrupt vectors remain mapped in the flash memory address space, the flash memory module will greatly increase the interrupt response time by adding several wait-states.

It is forbidden to enter in low power mode when the power-down mode is active.

28.4.8 Low power mode

The low power mode turns off most of the DC current sources within the flash memory module. Flash memory low power mode can be selected at ME_<mode>_MC.

The module (flash memory core and registers) is not accessible for read or write once it enters low power mode.

Wake-up time from low power mode is faster than wake-up time from power-down mode.

When exiting from low power mode the flash memory module returns to its pre-sleep state in all cases unless it is executing an erase high voltage operation at the time low power mode is entered.

If the flash memory module enters low power mode during an erase operation, MCR[ESUS] is programmed to '1'. The user may resume the erase operation at the time the module exits low power mode by programming MCR[ESUS] = 0. MCR[EHV] must be high to resume the erase operation.

If the flash memory module enters low power mode during a program operation, the operation will be in any case completed and the low power mode will be entered only after the programming end.

It is forbidden to enter power-down mode when the low power mode is active.

28.5 Register description

The CFlash and DFlash modules have respective sets of memory mapped registers. The CFlash register mapping is shown in *Table 322*. The DFlash register mapping is shown in *Table 323*.

Table 322. CFlash registers

Address offset	Register	Location
0x0000		*on page 28-639*
0x0004	CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML)	*on page 28-644*
0x0008	Reserved	
0x000C	CFlash Secondary Low/Mid Address Space Block Locking Register (CFLASH_SLL)	*on page 28-648*
0x0010	CFlash Low/Mid Address Space Block Select Register (CFLASH_LMS)	*on page 28-654*
0x0014	Reserved	
0x0018	CFlash Address Register (CFLASH_ADR)	*on page 28-655*
0x0028–0x0038	Reserved	
0x003C	CFlash User Test 0 register (CFLASH_UT0)	*on page 28-656*
0x0040	CFlash User Test 1 register (CFLASH_UT1)	*on page 28-658*
0x0044	CFlash User Test 2 register (CFLASH_UT2)	*on page 28-659*
0x0048	CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0)	*on page 28-660*

Table 322. CFlash registers (continued)

Address offset	Register	Location
0x004C	CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1)	on page 28-660
0x0050	CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2)	on page 28-661
0x0054	CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3)	on page 28-662
0x0058	CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4)	on page 28-663

Table 323. DFlash registers

Address offset	Register name	Location
0x0000	DFlash Module Configuration Register (DFLASH_MCR)	on page 28-668
0x0004	DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML)	on page 28-674
0x0008	Reserved	—
0x000C	DFlash Secondary Low/Mid Address Space Block Locking Register (DFLASH_SLL)	on page 28-678
0x0010	DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS)	on page 28-682
0x0014	Reserved	—
0x0018	DFlash Address Register (DFLASH_ADR)	on page 28-682
0x001C–0x0038	Reserved	—
0x003C	DFlash User Test 0 register (DFLASH_UT0)	on page 28-683
0x0040	DFlash User Test 1 register (DFLASH_UT1)	on page 28-686
0x0044	DFlash User Test 2 register (DFLASH_UT2)	on page 28-686
0x0048	DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0)	on page 28-687
0x004C	DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1)	on page 28-688
0x0050	DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2)	on page 28-689
0x0054	DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3)	on page 28-690
0x0058	DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4)	on page 28-691

In the following some nonvolatile 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 memory initialization phase, the FPEC reads these nonvolatile registers and updates the corresponding 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, embedded firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, censorship, platform flash memory controller, ...), the volatile registers are filled with all ‘1’s and the flash memory initialization ends setting low the PEG bit of the corresponding MCR (CFLASH_MCR or DFLASH_MCR).

28.5.1 CFlash register description

CFlash Module Configuration Register (CFLASH_MCR)

The CFlash Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Figure 348. CFlash Module Configuration Register (CFLASH_MCR)

Offset: 0x0000 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W	w1c															
Reset	0	0	0	0	0	0	1	0	0	0	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	w1c	w1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Table 324. CFLASH_MCR field descriptions

Field	Description
EDC	*Ecc Data Correction* EDC provides information on previous reads. If an ECC Single Error detection and correction occurred, 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 an ECC Double Error detection, this bit will not be 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. 0: Reads are occurring normally. 1: An ECC Single Error occurred and was corrected during a previous read.
SIZE	*array space SIZE* The value of SIZE field is dependent upon the size of the flash memory module; see *Table 325*.

Table 324. CFLASH_MCR field descriptions (continued)

Field	Description
LAS	*Low Address Space* The value of the LAS field corresponds to the configuration of the Low Address Space; see *Table 326*.
MAS	*Mid Address Space* The value of the MAS field corresponds to the configuration of the Mid Address Space; see *Table 327*.
EER	*Ecc event ERror* EER provides information on previous reads. If an ECC Double Error detection occurred, 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 an 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. 0: Reads are occurring normally. 1: An ECC Double Error occurred during a previous read.
RWE	*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 memory Matrix 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. 0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.
PEAS	*Program/Erase Access Space* PEAS is used to indicate 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 memory 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.

Table 324. CFLASH_MCR field descriptions (continued)

Field	Description
DONE	*modify operation DONE* DONE indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory 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 tPABT or tEABT, equal to P/E Abort Latency) after a 1 to 0 transition of EHV, which aborts a high voltage Program/Erase operation. DONE is set to 1 (within tESUS, time equals to Erase Suspend Latency) after a 0 to 1 transition of ESUS, which suspends an erase operation. 0: Flash memory is executing a high voltage operation. 1: Flash memory is not executing a high voltage operation.
PEG	*Program/Erase Good* The PEG bit indicates the completion status of the last flash memory 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 will cause PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the flash memory module is reset, unless a flash memory 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 an abort 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 will be PEG=1, indicating that the operation was succesful, 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 Read 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 UMIRS0-1. Aborting an Array Integrity Check or a Margin Read operation will cause PEG to be cleared to 0, indicating the sequence failed. 0: Program, Erase operation failed or Program, Erase, Array Integrity Check or Maring Mode aborted. 1: Program or Erase operation succesful or Array Integrity Check or Maring Mode completed.
PGM	*ProGraM* PGM is used to set up the flash memory 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 memory is not executing a Program sequence. 1: Flash memory is executing a Program sequence.
PSUS	*Program SUSpend* Write this bit has no effect, but the written data can be read back.

Table 324. CFLASH_MCR field descriptions (continued)

Field	Description
ERS	*ERaSe* ERS is used to set up the flash memory 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 memory is not executing an erase sequence. 1: Flash memory is executing an erase sequence.
ESUS	*Erase SUSpend* ESUS is used to indicate that the flash memory module is in Erase Suspend or in the process of entering a Suspend state. The flash memory module is in Erase Suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash memory in Erase Suspend. The flash memory module enters Suspend within t_{ESUS} of this transition. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to Erase. The flash memory 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	*Enable High Voltage* The EHV bit enables the flash memory 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 aborted, there is a 1 to 0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. Aborting a high voltage operation will leave the flash memory 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 memory is not enabled to perform an high voltage operation. 1: Flash memory is enabled to perform an high voltage operation.

Table 325. Array space size

SIZE	Array space size
000	128 KB
001	256 KB
010	512 KB
011	1024 KB
100	1536 KB
101	Reserved (2048 KB)
110	64 KB
111	Reserved

Table 326. Low address space configuration

LAS	Low address space sectorization
000	Reserved
001	Reserved
010	32 KB + 2 x 16 KB + 2 x 32 KB + 128 KB
011	Reserved
100	Reserved
101	Reserved
110	4 x 16 KB
111	Reserved

Table 327. Mid address space configuration

MAS	Mid address space sectorization
0	2 x 128 KB or 0 KB
1	Reserved

A number of CFLASH_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 memory 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. The bit changing priorities are detailed in *Table 328*.

Table 328. CFLASH_MCR bits set/clear priority levels

Priority level	CFLASH_MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more CFLASH_MCR bits simultaneously then only the bit with the lowest priority level is written.

If Stall/Abort-While-Write is enabled and an erase operation is started on one sector while fetching code from another then the following sequence is executed:

- CPU is stalled when flash is unavailable
- PEG flag set (stall case) or reset (abort case)
- Interrupt triggered if enabled

If Stall/Abort-While-Write is used then application software should ignore the setting of the RWE flag. The RWE flag should be cleared after each HV operation.

If Stall/Abort-While-Write is not used the application software should handle RWE error. See *Section 28.8.10, Read-while-write functionality*.

CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML)

The CFlash Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the CFLASH_SLL register, determine if the block is locked from Program or Erase. An “OR” of CFLASH_LML and CFLASH_SLL determine the final lock status.

Figure 349. CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML)

Offset: 0x0004 Access: 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	MLK	
W																
Reset	Defined by CFLASH_NVLML at CFlash Test Sector Address 0x403DE8. This location is user OTP (One Time Programmable). The CFLASH_NVLML register influences only the R/W bits of the CFLASH_LML register.															

	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	LLK					
W																
Reset	Defined by CFLASH_NVLML at CFlash Test Sector Address 0x403DE8. This location is user OTP (One Time Programmable). The CFLASH_NVLML register influences only the R/W bits of the CFLASH_LML register.															

Table 329. CFLASH_LML field descriptions

Field	Description
LME	*Low/Mid address space block Enable* This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) 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 will be 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 CFLASH_LML register. 0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written. 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.
TSLK	*Test/Shadow address space block LocK* This bit is used 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 TSLK register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until CFLASH_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 (also if CFLASH_SLL[STSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.

Table 329. CFLASH_LML field descriptions (continued)

Field	Description
MLK	*Mid address space block LocK* This field is used to lock the blocks of Mid Address Space from Program and Erase. MLK is related to sectors B0F7-6, respectively. A value of 1 in a bit of the MLK field signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the MLK field signifies that the corresponding block is available to receive program and erase pulses. The MLK field is not writable after an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK field is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the MLK field. The MLK field may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the MLK field (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the MLK field 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. MLK is not writable unless LME is high. 0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_SLL[SMLK] = 0). 1: Mid Address Space Block is locked and cannot be modified.
LLK	*Low address space block LocK* This field is used to lock the blocks of Low Address Space from Program and Erase. LLK[5:0] are related to sectors B0F5-0, respectively. LLK[15:6] are not used for this memory cut. A value of 1 in a bit of the LLK field signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the LLK field signifies that the corresponding block is available to receive program and erase pulses. The LLK field is not writable after an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK field is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK field. The LLK field may be written as a register. Reset will cause the field to go back to its TestFlash block value. The default value of the LLK field (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK field 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. Bits LLK[15:6] 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 (also if CFLASH_SLL[SLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

CFlash Nonvolatile Low/Mid Address Space Block Locking Register (CFLASH_NVLML)

The CFLASH_LML register has a related CFlash Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for CFLASH_LML. During the reset phase of the flash memory module, the CFLASH_NVLML register content is read and loaded into the CFLASH_LML.

The CFLASH_NVLML register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Figure 350. CFlash Nonvolatile Low/Mid address space block Locking register (CFLASH_NVLML)

Offset: 0x403DE8 Access: Read/write

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

Table 330. CFLASH_NVLML field descriptions

Field	Description
LME	*Low/Mid address space block Enable* This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) 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 will be 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 CFLASH_LML register. 0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written. 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.
TSLK	*Test/Shadow address space block LocK* This bit is used 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 TSLK register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until CFLASH_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 (also if CFLASH_SLL[STSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.

Table 330. CFLASH_NVLML field descriptions (continued)

Field	Description
MLK	*Mid address space block LocK* These bits are used to lock the blocks of Mid Address Space from Program and Erase. MLK[1:0] are related to sectors B0F7-6, respectively. A value of 1 in a bit of the MLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the MLK register signifies that the corresponding block is available to receive program and erase pulses. The MLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the MLK registers. The MLK 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 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 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. MLK is not writable unless LME is high. 0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_SLL[SMLK] = 0). 1: Mid Address Space Block is locked and cannot be modified.
LLK	*Low address space block LocK* These bits are used to lock the blocks of Low Address Space from Program and Erase. LLK[5:0] are related to sectors B0F5-0, respectively. LLK[15:6] are not used for this memory cut. A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses. The LLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK registers. 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 will always be 1 (independent of the TestFlash block), and register writes will have no effect. Bits LLK[15:6] 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 (also if CFLASH_SLL[SLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

CFlash Secondary Low/Mid Address Space Block Locking Register (CFLASH_SLL)

The CFlash 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 CFLASH_LML register, determine if the block is locked from Program or Erase. An “OR” of CFLASH_LML and CFLASH_SLL determine the final lock status.

Figure 351. CFlash Secondary Low/mid address space block Locking Register (CFLASH_SLL)

Offset: 0x000C Access: 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	STSLK	0	0	SMK	
W																
Reset	Defined by CFLASH_NVSLL at CFlash Test Sector Address 0x403DF8. This location is user OTP (One Time Programmable). The CFLASH_NVSLL register influences only the R/W bits of the CFLASH_SLL register.															

	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	SLK					
W																
Reset	Defined by CFLASH_NVSLL at CFlash Test Sector Address 0x403DF8. This location is user OTP (One Time Programmable). The CFLASH_NVSLL register influences only the R/W bits of the CFLASH_SLL register.															

Table 331. CFLASH_SLL field descriptions

Field	Description
SLE	*Secondary Low/mid address space block Enable* This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) 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 will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the CFLASH_SLL register. 0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written. 1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.
STSLK	*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 register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until CFLASH_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 (also if CFLASH_LML[TSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.

Table 331. CFLASH_SLL field descriptions (continued)

Field	Description
SMK	*Secondary Mid address space block locK* 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 B0F7-6, respectively. 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 CFLASH_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 will have no effect. SMK is not writable unless SLE is high. 0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_LML[MLK] = 0). 1: Mid Address Space Block is locked and cannot be modified.
SLK	*Secondary Low address space block locK* These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase. SLK[5:0] are related to sectors B0F5-0, respectively. SLK[15:6] are not used for this memory cut. 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 CFLASH_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 will cause 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 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. Bits SLK[15:6] 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 (also if CFLASH_LML[LLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

CFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register (CFLASH_NVSLL)

The CFLASH_SLL register has a related Nonvolatile Secondary Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for SLL. During the reset phase of the flash memory module, the CFLASH_NVSLL register content is read and loaded into the CFLASH_SLL.

The CFLASH_NVSLL register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Figure 352. CFlash Nonvolatile Secondary Low/mid address space block Locking register (CFLASH_NVSLL)

Offset: 0x403DF8 Access: Read/write

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

Table 332. CFLASH_NVSLL field descriptions

Field	Description
SLE	*Secondary Low/mid address space block Enable* This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) 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 will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the CFLASH_SLL register. 0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written. 1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.
STSLK	*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 register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until CFLASH_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 (also if CFLASH_LML[TSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.

Table 332. CFLASH_NVSLL field descriptions (continued)

Field	Description
SMK	*Secondary Mid address space block locK* 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 B0F7-6, respectively. 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 CFLASH_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 will have no effect. SMK is not writable unless SLE is high. 0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_LML[MLK] = 0). 1: Mid Address Space Block is locked and cannot be modified.
SLK	*Secondary Low address space block locK* These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase. SLK[5:0] are related to sectors B0F5-0, respectively. SLK[15:6] are not used for this memory cut. 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 CFLASH_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 will cause 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 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. Bits SLK[15:6] 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 (also if CFLASH_LML[LLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

CFlash Low/Mid Address Space Block Select Register (CFLASH_LMS)

Figure 353. CFlash Low/Mid address space block Select register (CFLASH_LMS)

Offset: 0x00010 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	MSL	
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	LSL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The CFLASH_LMS register provides a means to select blocks to be operated on during erase.

Table 333. CFLASH_LMS field descriptions

Field	Description
MSL	*Mid address space block SeLect* 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. MSL[1:0] are related to sectors B0F7-6, 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 will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. 0: Mid Address Space Block is unselected for erase. 1: Mid Address Space Block is selected for erase.
LSL	*Low address space block SeLect* 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. LSL[5:0] are related to sectors B0F5-0, respectively. LSL[15:6] are not used for this memory cut. 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. Bits LSL[15:6] 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.

CFlash Address Register (CFLASH_ADR)

Figure 354. CFlash Address Register (CFLASH_ADR)

Offset: 0x00018 Access: 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	AD22	AD21	AD20	AD19	AD18	AD17	AD16
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	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The CFLASH_ADR provides the first failing address in the event module failures (ECC or FPEC) occur or the first address at which an ECC single error correction occurs.

Table 334. CFLASH_ADR field descriptions

Field	Description
AD	*ADdress 22-3* (Read Only) The Address Register provides the first failing address in the event of ECC error (CFLASH_MCR[EER] = 1) or the first failing address in the event of RWW error (CFLASH_MCR[RWE] = 1), or the address of a failure that may have occurred in a FPEC operation (CFLASH_MCR[PEG] = 0). The Address Register also provides the first address at which an ECC single error correction occurs (CFLASH_MCR[EDC] = 1). The ECC double error detection takes the highest priority, followed by the FPEC error and the ECC single error correction. When accessed CFLASH_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 335*. 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.

Table 335. CFLASH_ADR content: priority list

Priority level	Error flag	CFLASH_ADR content
1	CFLASH_MCR[EER] = 1	Address of first ECC Double Error
2	CFLASH_MCR[RWE] = 1	Address of first RWW Error
3	CFLASH_MCR[PEG] = 0	Address of first FPEC Error
4	CFLASH_MCR[EDC] = 1	Address of first ECC Single Error Correction

CFlash User Test 0 register (CFLASH_UT0)

The User Test Registers provide the user with the ability to test features on the flash memory module. The User Test 0 Register allows to control the way in which the flash memory content check is done.

Bits MRE, MRV, AIS, EIE and DSI[7:0] of the User Test 0 Register are not accessible whenever CFLASH_MCR[DONE] or UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 355. CFlash User Test 0 register (CFLASH_UT0)

Offset: 0x0003C Access: 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	DSI							
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 336. CFLASH_UT0 field descriptions

Field	Description
UTE	*User Test Enable* This status bit gives indication when User Test is enabled. All bits in CFLASH_UT0-2 and CFLASH_UMISR0-4 are locked when this bit 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 0xF9F99999 must be written to the CFLASH_UT0 register.
DSI	*Data Syndrome Input* These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. Bits DSI[7:0] correspond to the 8 syndrome bits on a double word. These bits are not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: The syndrome bit is forced at 0. 1: The syndrome bit is forced at 1.
X	*Reserved* This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.
MRE	*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 inside the Array Integrity Checks sequences. Margin reads are only active during Array Integrity Checks; Normal User reads are not affected by MRE. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: Margin reads are not enabled 1: Margin reads are enabled.

Table 336. CFLASH_UT0 field descriptions (continued)

Field	Description
MRV	*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 CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has 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	*ECC data Input Enable* EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: ECC Logic Check is not enabled. 1: ECC Logic Check is enabled.
AIS	*Array Integrity Sequence* AIS determines the address sequence to be used during array integrity checks or Margin Read . 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. The usage of proprietary sequence is forbidden in Margin Read. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: Array Integrity sequence is proprietary sequence. 1: Array Integrity or f sequence is sequential.
AIE	*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 (CFLASH_UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if CFLASH_MCR[ERS], CFLASH_MCR[PGM] and CFLASH_MCR[EHV] are all low. 0: Array Integrity Checks, Margin Read and ECC Logic Checks are not enabled. 1: Array Integrity Checks, Margin Read and ECC Logic Checks are enabled.
AID	*Array Integrity Done* AID will be cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID will be set to indicate that the Array Integrity Check is complete. At this time the MISR (CFLASH_UMISR0-4) can be checked. 0: Array Integrity Check is on-going. 1: Array Integrity Check is done.

CFlash User Test 1 register (CFLASH_UT1)

The CFLASH_UT1 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 CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 356. CFlash User Test 1 register (CFLASH_UT1)

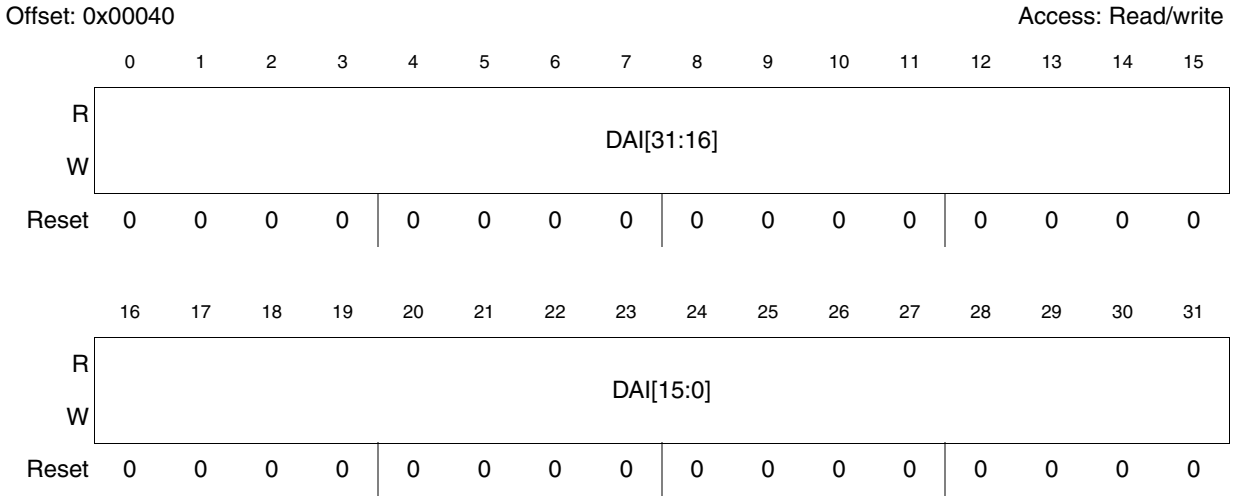


Table 337. CFLASH_UT1 field descriptions

Field	Description
DAI[31:0]	*Data Array Input, bits 31–0* These bits represent the input of even word of ECC logic used in the ECC Logic Check. Bits DAI[31:00] 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.

CFlash User Test 2 register (CFLASH_UT2)

The CFLASH_UT2 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 CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 357. CFlash User Test 2 register (CFLASH_UT2)

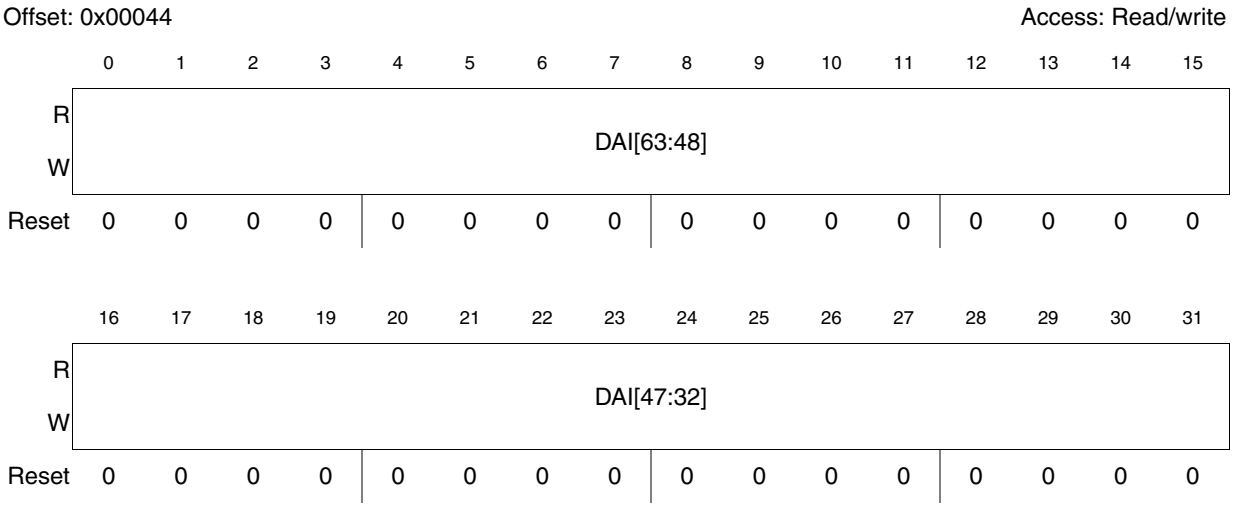


Table 338. CFLASH_UT2 field descriptions

Field	Description
DAI[63:32]	*Data Array Input, bits 63–32* These bits represent the input of odd word of ECC logic used in the ECC Logic Check. Bits DAI[63:32] 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.

CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0)

The CFLASH_UMISR0 register provides a mean to evaluate the Array Integrity.

The User Multiple Input Signature Register 0 represents the bits 31:0 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH_UMISR0 Register is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 358. CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0)

Offset: 0x00048 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS0[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	MS0[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 339. CFLASH_UMISR0 field descriptions

Field	Description
MS0[31:0]	*Multiple input Signature, bits 31–0* These bits represent the MISR value obtained 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 CFLASH_UMISR0 register.

CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1)

The CFLASH_UMISR1 provides a means to evaluate the Array Integrity.

The CFLASH_UMISR1 represents the bits 63:32 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH_UMISR1 is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 359. CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1)

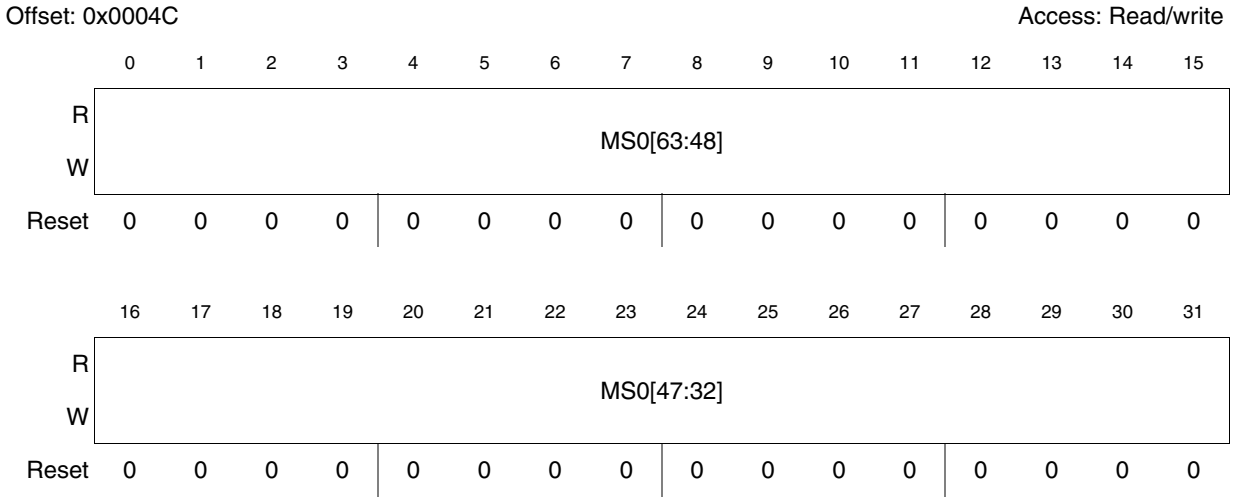


Table 340. CFLASH_UMISR1 field descriptions

Field	Description
MS0[63:32]	*Multiple input Signature, bits 63–32* 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 CFLASH_UMISR1.

CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2)

The CFLASH_UMISR2 provides a means to evaluate the Array Integrity.

The CFLASH_UMISR2 represents the bits 95:64 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH_UMISR2 is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 360. CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2)

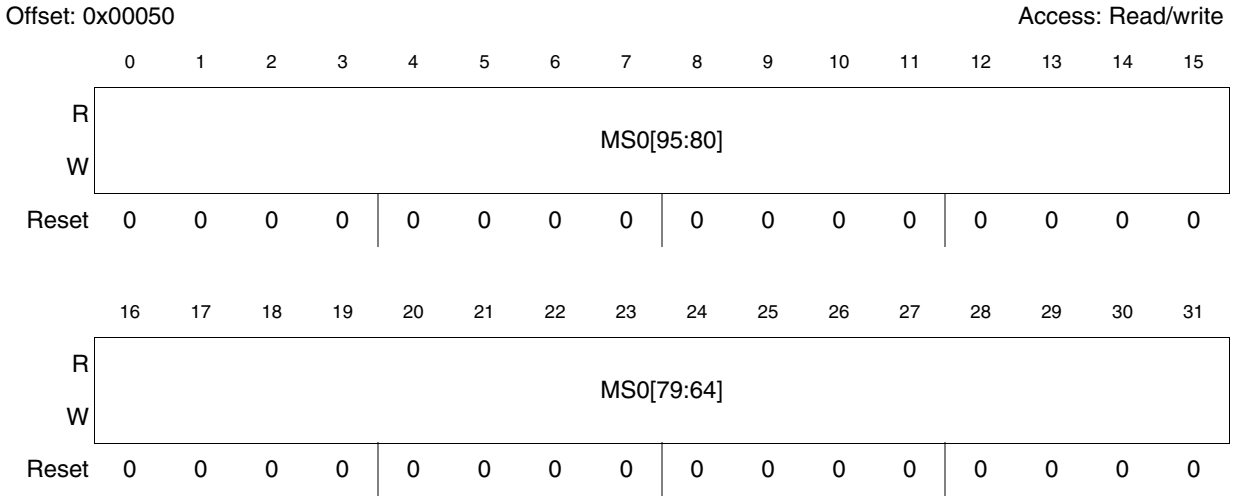


Table 341. CFLASH_UMISR2 field descriptions

Field	Description
MS0[95:64]	*Multiple input Signature, bits 95–64* These bits represent the MISR value obtained 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 CFLASH_UMISR2.

CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3)

The CFLASH_UMISR3 provides a mean to evaluate the Array Integrity.

The CFLASH_UMISR3 represents the bits 127:96 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH_UMISR3 is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 361. CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3)

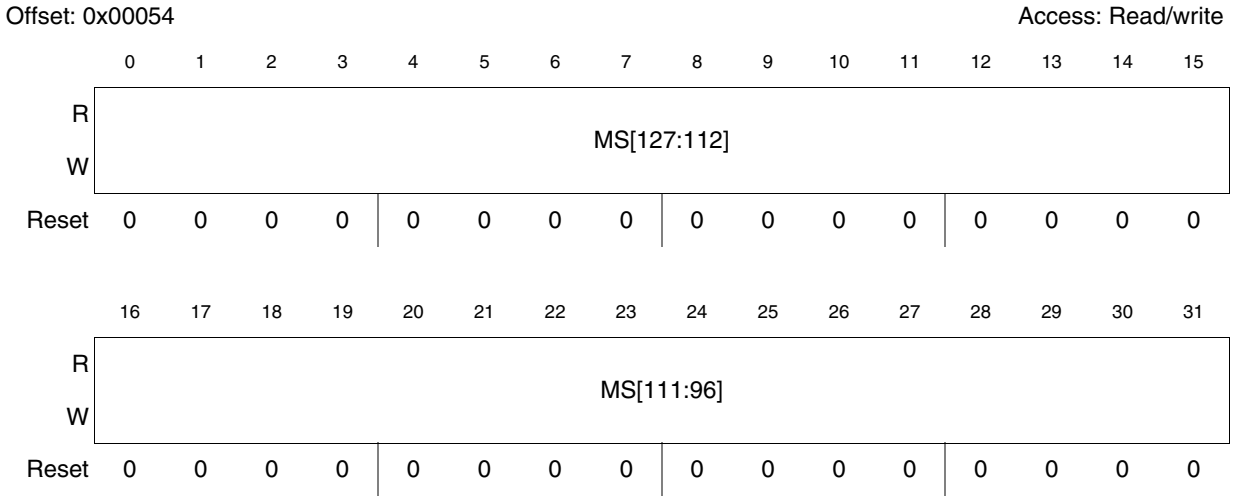


Table 342. CFLASH_UMISR3 field descriptions

Field	Description
MS[127:96]	*Multiple input Signature, bits127–96* These bits represent the MISR value obtained accumulating the bits 127:96 of all the pages read from the flash memory. The MS can be seeded to any value by writing the CFLASH_UMISR3.

CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4)

The CFLASH_UMISR4 provides a mean to evaluate the Array Integrity.

The CFLASH_UMISR4 represents the ECC bits of the whole 144 bits 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 MISR are respectively the double and single ECC error detection for odd and even Double Word.

The CFLASH_UMISR4 is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 362. CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4)

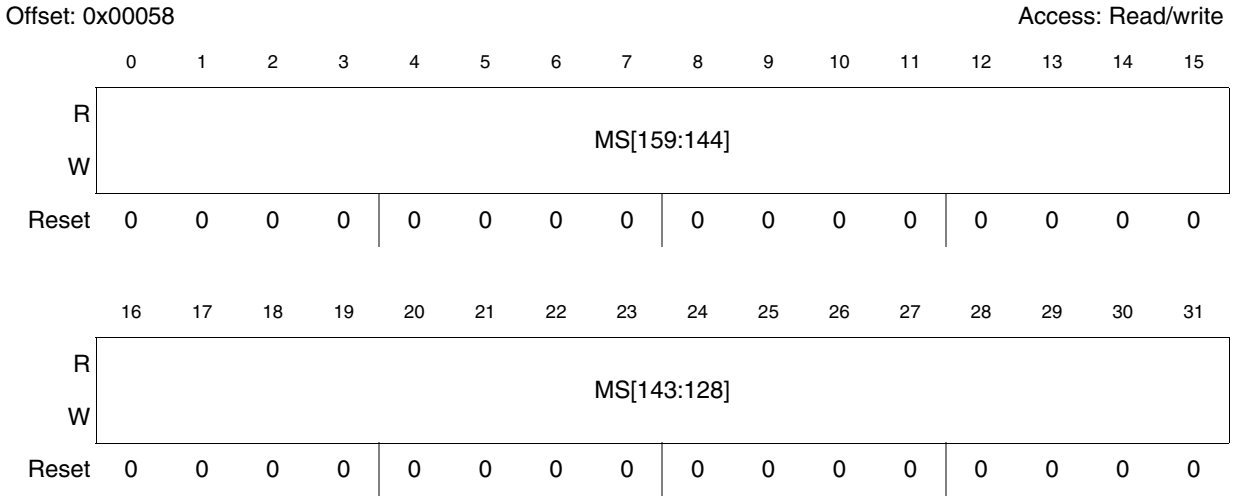


Table 343. CFLASH_UMISR4 field descriptions

Field	Description
MS[159:128]	*Multiple input Signature, bits 159–128* These bits represent the MISR value obtained accumulating: the 8 ECC bits for the even Double Word (on MS[135:128]); the single ECC error detection for even Double Word (on MS138); the double ECC error detection for even Double Word (on MS139); the 8 ECC bits for the odd Double Word (on MS[151:144]); the single ECC error detection for odd Double Word (on MS154); the double ECC error detection for odd Double Word (on MS155). The MS can be seeded to any value by writing the CFLASH_UMISR4 register.

CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0)

The nonvolatile private censorship password 0 register contains the 32 LSB of the Password used to validate the Censorship information contained in NVSCC0–1 registers.

Figure 363. CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0)

Offset: 0x203DD8 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD[31:16]															
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 W	PWD[15:0]															
Reset	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0

Table 344. NVPWD0 field descriptions

Field	Description
PWD[31:0]	*Password, bits 31–0* These bits represent the 32 LSB of the Private Censorship Password.

CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1)

The nonvolatile private censorship password 1 register contains the 32 MSB of the Password used to validate the Censorship information contained in NVSCC0–1 registers.

Figure 364. CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1)

Offset: 0x203DDC Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD[63:48]															
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 W	PWD[47:32]															
Reset	1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1

Table 345. NVPWD1 field descriptions

Field	Description
PWD[63:32]	*Password, bits 63–32* These bits represent the 32 MSB of the Private Censorship Password.

Note: In a secured device, starting with a serial boot, it is possible to read the content of the four flash locations where the RCHW can be stored. For example if the RCHW is stored at address 0x00000000, the reads at address 0x00000000, 0x00000004, 0x00000008 and 0x0000000C will return a correct value. Any other flash address cannot be accessed.

CFlash Nonvolatile System Censorship Control 0 register (NVSCC0)

The NVSCC0 register stores the 32 LSB of the Censorship Control Word of the device.

The NVSCC0 is a nonvolatile register located in the Shadow sector: it is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user.

Figure 365. CFlash Nonvolatile System Censorship Control 0 register (NVSCC0)

Offset: 0x203DE0 Access: Read/write

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

Table 346. NVSCC0 field descriptions

Field	Description
SC[15:0]	*Serial Censorship control word, bits 15-0* These bits represent the 16 LSB of the Serial Censorship Control Word (SCCW). If SC15-0 = 0x55AA and NVSCC1 = NVSCC0 the Public Access is disabled. If SC15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Public Access is enabled.
CW[15:0]	*Censorship control Word, bits 15-0* These bits represent the 16 LSB of the Censorship Control Word (CCW). If CW15-0 = 0x55AA and NVSCC1 = NVSCC0 the Censored Mode is disabled. If CW15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Censored Mode is enabled.

CFlash Nonvolatile System Censorship Control 1 register (NVSCC1)

The NVSCC1 register stores the 32 MSB of the Censorship Control Word of the device.

The NVSCC1 is a nonvolatile register located in the Shadow sector: it is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user.

Figure 366. CFlash Nonvolatile System Censorship Control 1 register (NVSCC1)

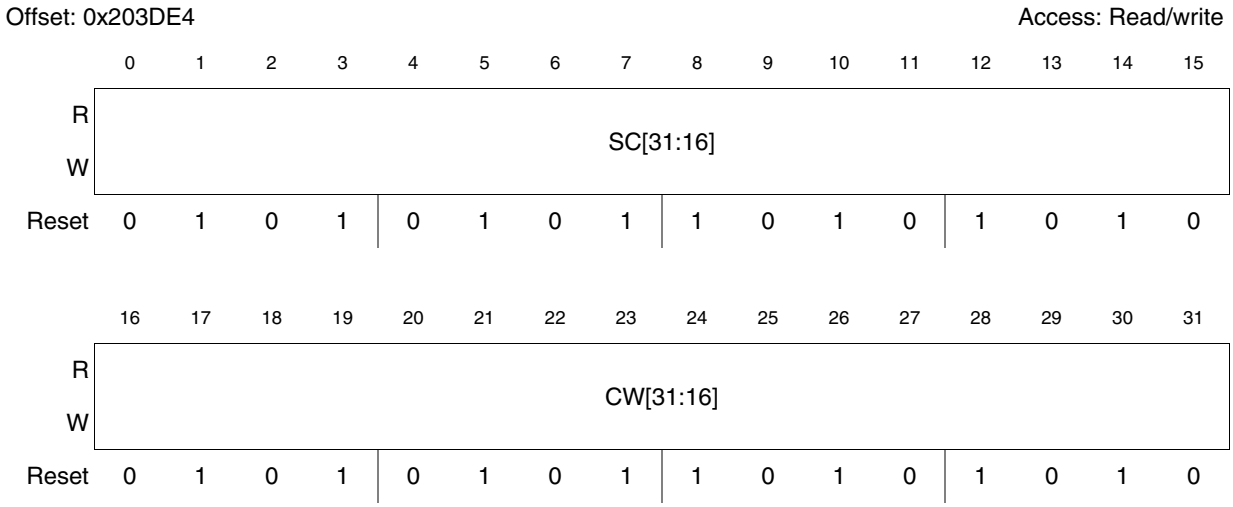


Table 347. NVSCC1 field descriptions

Field	Description
SC[31:16]	*Serial Censorship control word, bits 31-16* These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW). If SC15-0 = 0x55AA and NVSCC1 = NVSCC0 the Public Access is disabled. If SC15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Public Access is enabled.
CW[31:16]	*Censorship control Word, bits 31-16* These bits represent the 16 MSB of the Censorship Control Word (CCW). If CW15-0 = 0x55AA and NVSCC1 = NVSCC0 the Censored Mode is disabled. If CW15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Censored Mode is enabled.

CFlash Nonvolatile User Options register (NVUSRO)

The nonvolatile User Options Register contains configuration information for the user application.

The NVUSRO register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Figure 367. CFlash Nonvolatile User Options register (NVUSRO)

Offset: 0x203E18 Access: Read/write

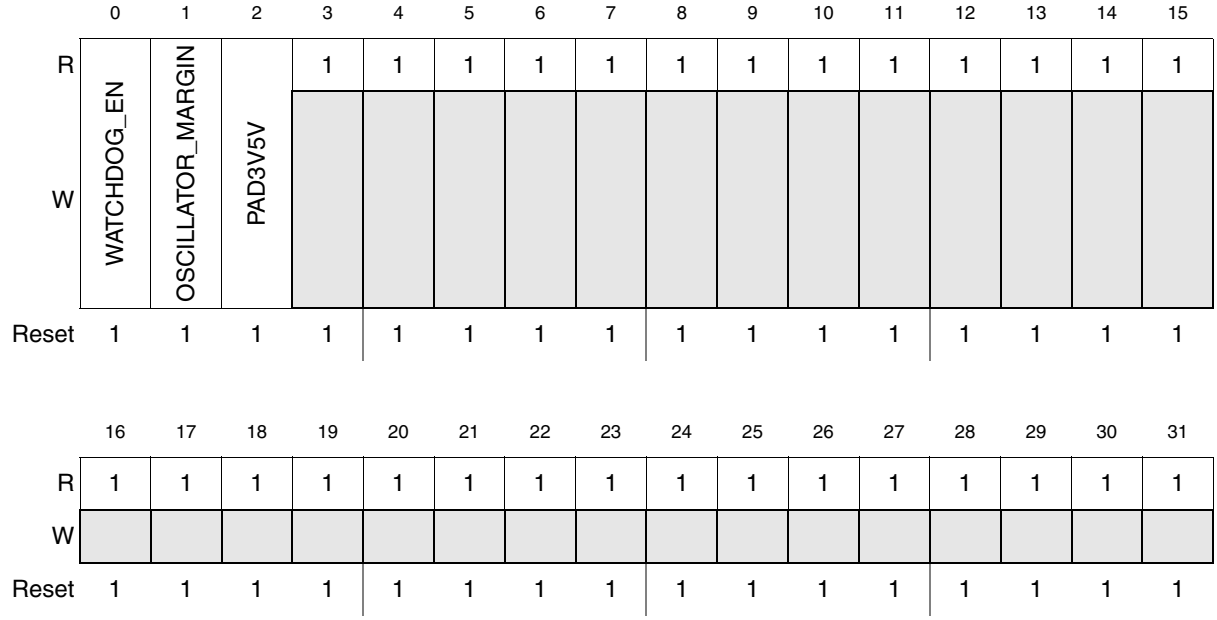


Table 348. NVUSRO field descriptions

Field	Description
WATCHDOG_EN	*WATCHDOG_EN* 0: Disable after reset 1: Enable after reset Default manufacturing value before flash memory initialization is ‘1’
OSCILLATOR_ MARGIN	*OSCILLATOR_MARGIN* 0: Low consumption configuration (4 MHz/8 MHz) 1: High margin configuration (4 MHz/16 MHz) Default manufacturing value before flash memory initialization is ‘1’
PAD3V5V	*PAD3V5V* 0: High voltage supply is 5.0 V 1: High voltage supply is 3.3 V Default manufacturing value before flash memory initialization is ‘1’ (3.3 V) which should ensure correct minimum slope for boundary scan.

28.5.2 DFlash register description

DFlash Module Configuration Register (DFLASH_MCR)

The Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Figure 368. DFlash Module Configuration Register (DFLASH_MCR)

Address offset: 0x0000 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W	w1c															
Reset	0	0	0	0	0	1	1	0	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	w1c	w1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Table 349. DFLASH_MCR field descriptions

Field	Description
EDC	*ECC Data Correction* EDC provides information on previous reads. If an ECC Single Error detection and correction occurred, 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 an ECC Double Error detection, this bit will not be 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. The function of this bit is device dependent and it can be configured to be disabled. 0: Reads are occurring normally. 1: An ECC Single Error occurred and was corrected during a previous read.
SIZE	*array space SIZE* The value of SIZE field is dependent upon the size of the flash memory module; see *Table 350*.
LAS	*Low Address Space* The value of the LAS field corresponds to the configuration of the Low Address Space; see *Table 351*.
MAS	*Mid Address Space* The value of the MAS field corresponds to the configuration of the Mid Address Space; see *Table 352*.
EER	*ECC event Error* EER provides information on previous reads. If an ECC Double Error detection occurred, 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 an 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. 0: Reads are occurring normally. 1: An ECC Double Error occurred during a previous read.

Table 349. DFLASH_MCR field descriptions (continued)

Field	Description
RWE	*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 will be set to '1'. Read-While-Write Error means that a read access to the flash memory Matrix 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. 0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.
PEAS	*Program/Erase Access Space* PEAS is used to indicate 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 memory 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	*modify operation DONE* DONE indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory 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 aborts a high voltage Program/Erase operation. DONE is set to 1 (within t_{ESUS}, time equals to Erase Suspend Latency) after a 0 to 1 transition of ESUS, which suspends an erase operation. 0: Flash memory is executing a high voltage operation. 1: Flash memory is not executing a high voltage operation.

Table 349. DFLASH_MCR field descriptions (continued)

Field	Description
PEG	*Program/Erase Good* The PEG bit indicates the completion status of the last flash memory 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 will cause PEG to be cleared to '0', indicating the sequence failed. PEG is set to '1' when the flash memory module is reset, unless a flash memory 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 an abort 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 will be 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 Read 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 UMIRS0-1. Aborting an Array Integrity Check or a Margin Read operation will cause PEG to be cleared to 0, indicating the sequence failed. 0: Program, Erase operation failed or Program, Erase, Array Integrity Check or Maring Mode aborted. 1: Program or Erase operation succesful or Array Integrity Check or Maring Mode completed.
PGM	*ProGraM* PGM is used to set up the flash memory 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 DFLASH_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 memory is not executing a Program sequence. 1: Flash memory is executing a Program sequence.
PSUS	**PSUS:** *Program SUSpend* Write this bit has no effect, but the written data can be read back.
ERS	*ERaSe* ERS is used to set up the flash memory 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 DFLASH_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 memory is not executing an erase sequence. 1: Flash memory is executing an erase sequence.

Table 349. DFLASH_MCR field descriptions (continued)

Field	Description
ESUS	*Erase SUSpend* ESUS is used to indicate that the flash memory module is in Erase Suspend or in the process of entering a Suspend state. The flash memory module is in Erase Suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash memory in Erase Suspend. The flash memory module enters Suspend within t_{ESUS} of this transition. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to Erase. The flash memory 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	*Enable High Voltage* The EHV bit enables the flash memory 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, DFLASH_UT0[AIE] = 0) Program (ERS = 0, ESUS = 0, PGM = 1, DFLASH_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 aborted, there is a 1 to 0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. Aborting a high voltage operation will leave the flash memory 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 memory is not enabled to perform an high voltage operation. 1: Flash memory is enabled to perform an high voltage operation.

Table 350. Array space size

SIZE	Array space size
000	128 KB
001	256 KB
010	512 KB
011	Reserved (1024 KB)
100	Reserved (1536 KB)
101	Reserved (2048 KB)

Table 350. Array space size (continued)

SIZE	Array space size
110	64 KB
111	Reserved

Table 351. Low address space configuration

LAS	Low address space sectorization
000	Reserved
001	Reserved
010	32 KB + 2 x 16 KB + 2 x 32 KB + 128 KB
011	Reserved
100	Reserved
101	Reserved
110	4 x 16 KB
111	Reserved

Table 352. Mid address space configuration

MAS	Mid address space sectorization
0	2 x 128KB
1	Reserved

A number of DFLASH_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 memory 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. The bit changing priorities are detailed in the *Table 353*.

Table 353. DFLASH_MCR bits set/clear priority levels

Priority level	DFLASH_MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more DFLASH_MCR bits simultaneously then only the bit with the lowest priority level is written.

If Stall/Abort-While-Write is enabled and an erase operation is started on one sector while fetching code from another then the following sequence is executed:

- CPU is stalled when flash is unavailable
- PEG flag set (stall case) or reset (abort case)
- Interrupt triggered if enabled

If Stall/Abort-While-Write is used then application software should ignore the setting of the RWE flag. The RWE flag should be cleared after each HV operation.

If Stall/Abort-While-Write is not used the application software should handle RWE error. See *Section 28.8.10, Read-while-write functionality*.

DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML)

The DFlash Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the DFLASH_SLL register, determine if the block is locked from Program or Erase. An “OR” of DFLASH_LML and DFLASH_SLL determine the final lock status.

Figure 369. DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML)

Offset: 0x0004 Access: 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	0	0
W																
Reset	Defined by DFLASH_NVLML at DFlash Test Sector Address 0xC03DE8. This location is user OTP (One Time Programmable). The DFLASH_NVLML register influences only the R/W bits of the DFLASH_LML register.															

	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	LLK			
W																
Reset	Defined by DFLASH_NVLML at DFlash Test Sector Address 0xC03DE8. This location is user OTP (One Time Programmable). The DFLASH_NVLML register influences only the R/W bits of the DFLASH_LML register.															

Table 354. DFLASH_LML field descriptions

Field	Description
LME	*Low/Mid address space block Enable* This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) 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 will be 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 DFLASH_LML register. 0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written. 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.
TSLK	*Test/Shadow address space block LocK* This bit is used 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 TSLK register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until DFLASH_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 (also if DFLASH_SLL[STSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.
LLK	*Low address space block LocK* This field is used to lock the blocks of Low Address Space from Program and Erase. LLK[3:0] are related to sectors B1F3-0, respectively. LLK[15:4] are not used for this memory cut. A value of 1 in a bit of the LLK field signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the LLK field signifies that the corresponding block is available to receive program and erase pulses. The LLK field is not writable after an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK field is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK field. The LLK field may be written as a register. Reset will cause the field to go back to its TestFlash block value. The default value of the LLK field (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK field 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 64 KB flash memory module 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 (also if DFLASH_SLL[SLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

DFlash Nonvolatile Low/Mid Address Space Block Locking Register (DFLASH_NVLML)

The DFLASH_LML register has a related Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for DFLASH_LML. During the reset phase of the flash memory module, the DFLASH_NVLML register content is read and loaded into the DFLASH_LML.

The DFLASH_NVLML register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Figure 370. DFlash Nonvolatile Low/Mid address space block Locking register (DFLASH_NVLML)

Offset: 0xC03DE8 Access: Read/write

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

Table 355. DFLASH_NVLML field descriptions

Field	Description
LME	*Low/Mid address space block Enable* This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) 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 will be 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 DFLASH_LML register. 0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written. 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.
TSLK	*Test/Shadow address space block LocK* This bit is used 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 TSLK register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until DFLASH_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 (also if DFLASH_SLL[STSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.
LLK	*Low address space block LocK* These bits are used to lock the blocks of Low Address Space from Program and Erase. LLK[3:0] are related to sectors B1F3-0, respectively. LLK[15:4] are not used for this memory cut. A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses. The LLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK registers. 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 will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the 64 KB flash memory module 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 (also if DFLASH_SLL[SLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

DFlash Secondary Low/Mid Address Space Block Locking Register (DFLASH_SLL)

The DFlash 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 DFLASH_LML register, determine if the block is locked from Program or Erase. An “OR” of DFLASH_LML and DFLASH_SLL determine the final lock status.

Figure 371. DFlash Secondary Low/mid address space block Locking register (DFLASH_SLL)

Offset: 0x000C Access: 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	STSLK	0	0	0	0
W																
Reset	Defined by DFLASH_NVSLL at DFlash Test Sector Address 0xC03DF8. This location is user OTP (One Time Programmable). The DFLASH_NVSLL register influences only the R/W bits of the DFLASH_SLL register.															

	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	SLK			
W																
Reset	Defined by DFLASH_NVSLL at DFlash Test Sector Address 0xC03DF8. This location is user OTP (One Time Programmable). The DFLASH_NVSLL register influences only the R/W bits of the DFLASH_SLL register.															

Table 356. DFLASH_SLL field descriptions

Field	Description
SLE	*Secondary Low/mid address space block Enable* This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) 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 will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the DFLASH_SLL register. 0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written. 1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.
STSLK	*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 register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until DFLASH_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 (also if DFLASH_LML[TSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.
SLK	*Secondary Low address space block locK* These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase. SLK[3:0] are related to sectors B1F3-0, respectively. SLK[15:4] are not used for this memory cut. 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 DFLASH_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 will cause 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 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 64 KB flash memory module 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 (also if DFLASH_LML[LLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

DFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register (DFLASH_NVSLL)

The DFLASH_SLL register has a related Nonvolatile Secondary Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for DFLASH_SLL. During the reset phase of the flash memory module, the DFLASH_NVSLL register content is read and loaded into the DFLASH_SLL.

The DFLASH_NVSLL register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Figure 372. DFlash Nonvolatile Secondary Low/mid address space block Locking register (DFLASH_NVSLL)

Offset: 0xC03DF8 Access: Read/write

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

Table 357. DFLASH_NVSLL field descriptions

Field	Description
SLE	*Secondary Low/mid address space block Enable* This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) 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 will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the DFLASH_SLL register. 0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written. 1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.
STSLK	*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 register signifies that the Test/shadow sector is locked for Program and Erase. A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until DFLASH_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 (also if DFLASH_LML[TSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.
SLK	*Secondary Low address space block locK* These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase. SLK[3:0] are related to sectors B1F3-0, respectively. SLK[15:4] are not used for this memory cut. 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 DFLASH_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 will cause 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 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 64 KB flash memory module 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 (also if DFLASH_LML[LLK] = 0). 1: Low Address Space Block is locked and cannot be modified.

DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS)

The DFLASH_LMS register provides a means to select blocks to be operated on during erase.

Figure 373. DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS)

Offset: 0x00010 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	LSL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 358. DFLASH_LMS field descriptions

Field	Description
LSL	*Low address space block SeLect* 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. LSL[3:0] are related to sectors B1F3-0, respectively. LSL[15:4] are not used for this memory cut. 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 80 KB flash memory module 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.

DFlash Address Register (DFLASH_ADR)

The DFLASH_ADR provides the first failing address in the event module failures (ECC, RWW or FPEC) occur or the first address at which an ECC single error correction occurs.

Figure 374. DFlash Address Register (DFLASH_ADR)

Address offset: 0x00018 Access: 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	AD22	AD21	AD20	AD19	AD18	AD17	AD16
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	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 359. DFLASH_ADR field descriptions

Field	Description
AD[22:3]	*ADdress 22-3* The Address Register provides the first failing address in the event of ECC error (DFLASH_MCR[EER] set) or the first failing address in the event of RWW error (DFLASH_MCR[RWE] set), or the address of a failure that may have occurred in a FPEC operation (DFLASH_MCR[PEG] cleared). The Address Register also provides the first address at which an ECC single error correction occurs (DFLASH_MCR[EDC] set), if the device is configured to show this feature. 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 DFLASH_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 the *Table 360*. 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 the Address Register is read only.

Table 360. DFLASH_ADR content: priority list

Priority level	Error flag	DFLASH_ADR content
1	DFLASH_MCR[EER] = 1	Address of first ECC Double Error
2	DFLASH_MCR[RWE] = 1	Address of first RWW Error
3	DFLASH_MCR[PEG] = 0	Address of first FPEC Error
4	DFLASH_MCR[EDC] = 1	Address of first ECC Single Error Correction

DFlash User Test 0 register (DFLASH_UT0)

The User Test Registers provide the user with the ability to test features on the flash memory module.

The User Test 0 Register allows to control the way in which the flash memory content check is done.

Bits MRE, MRV, AIS, EIE and DSI[7:0] of the User Test 0 Register are not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 375. DFlash User Test 0 register (DFLASH_UT0)

Offset: 0x0003C Access: 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	DSI							
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	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 361. DFLASH_UT0 field descriptions

Field	Description
UTE	*User Test Enable* This status bit gives indication when User Test is enabled. All bits in DFLASH_UT0-2 and DFLASH_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 0xF9F99999 must be written to the DFLASH_UT0 register.
DSI	*Data Syndrome Input* These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. Bits DSI[7:0] correspond to the 8 syndrome bits on a double word. These bits are not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: The syndrome bit is forced at 0. 1: The syndrome bit is forced at 1.
X	*Reserved* This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Table 361. DFLASH_UT0 field descriptions (continued)

Field	Description
MRE	*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 DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: Margin reads are not enabled, all reads are User mode reads. 1: Margin reads are enabled.
MRV	*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 DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has 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	*ECC data Input Enable* EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: ECC Logic Check is not enabled. 1: ECC Logic Check is enabled.
AIS	Array Integrity Sequence AIS determines the address sequence to be used during array integrity checks or Margin Read. 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. Proprietary sequence is forbidden in Margin Read. 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 DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: Array Integrity equence is proprietary sequence. 1: Array Integrity or Margin Read sequence is sequential.
AIE	*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 (DFLASH_UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if DFLASH_MCR[ERS], DFLASH_MCR[PGM] and DFLASH_MCR[EHV] are all low. 0: Array Integrity Checks are not enabled. 1: Array Integrity Checks are enabled.
AID	*Array Integrity Done* AID will be cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID will be set to indicate that the Array Integrity Check is complete. At this time the MISR (DFLASH_UMISR0-4) can be checked. 0: Array Integrity Check is on-going. 1: Array Integrity Check is done.

DFlash User Test 1 register (DFLASH_UT1)

The DFLASH_UT1 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 DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 376. DFlash User Test 1 register (DFLASH_UT1)

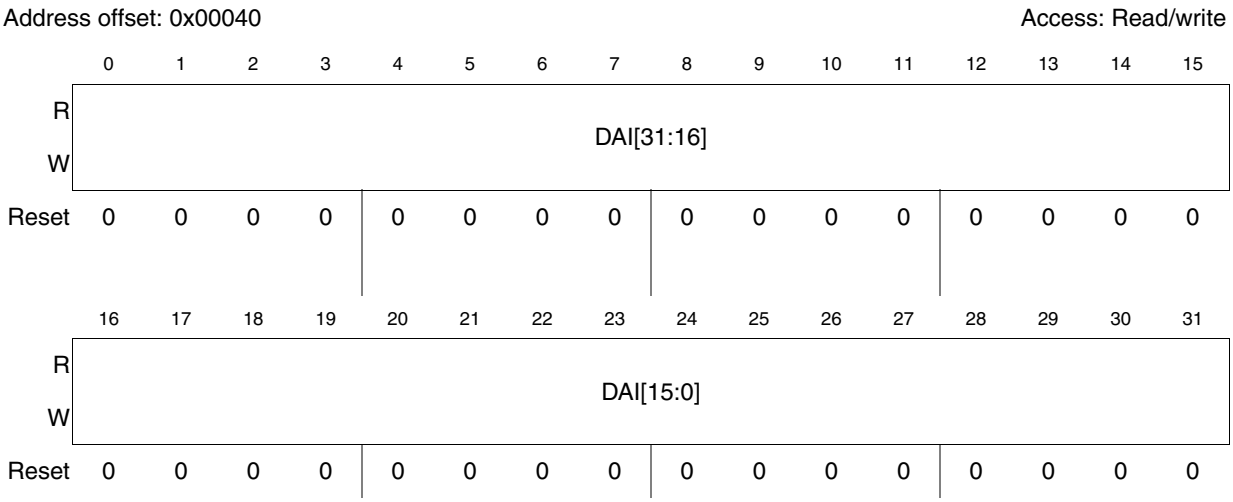


Table 362. DFLASH_UT1 field descriptions

Field	Description
DAI[31:16]	*Data Array Input, bits 31-0* These bits represent the input of even word of ECC logic used in the ECC Logic Check. Bits DAI[31:00] 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.

DFlash User Test 2 register (DFLASH_UT2)

The DFLASH_UT2 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 DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 377. DFlash User Test 2 register (DFLASH_UT2)

Offset: 0x00044 Reset value: 0x0000_0000

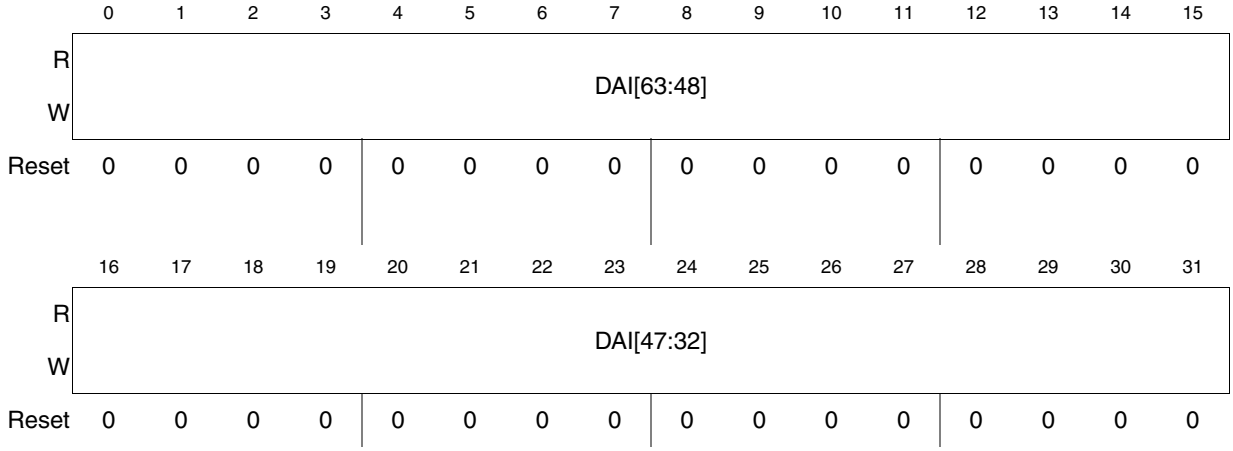


Table 363. DFLASH_UT2 field descriptions

Field	Description
DAI[63:32]	*Data Array Input, bits 63-32* These bits represent the input of odd word of ECC logic used in the ECC Logic Check. Bits DAI[63:32] 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.

DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0)

The DFLASH_UMISR0 provides a means to evaluate the Array Integrity.

The DFLASH_UMISR0 represents the bits 31:0 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH_UMISR0 is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 378. DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0)

Address offset: 0x00048 Reset value: 0x0000_0000

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

Table 364. DFLASH_UMISR0 field descriptions

Field	Description
MS[31:0]	*Multiple input Signature, bits 31–0* These bits represent the MISR value obtained 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 DFLASH_UMISR0 register.

DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1)

The DFLASH_UMISR1 provides a mean to evaluate the Array Integrity.

The DFLASH_UMISR1 represents the bits 63:32 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH_UMISR1 is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 379. DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1)

Address offset: 0x0004C Reset value: 0x0000_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS[63:48]															
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	MS[47:32]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 365. DFLASH_UMISR1 field descriptions

Field	Description
MS[63:32]	*Multiple input Signature, bits 63-32* 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 DFLASH_UMISR1.

DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2)

The DFLASH_UMISR2 provides a mean to evaluate the Array Integrity.

The DFLASH_UMISR2 represents the bits 95:64 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH_UMISR2 is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 380. DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2)

Address offset: 0x00050 Reset value: 0x0000_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS[95:80]															
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[79:64]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 366. DFLASH_UMISR2 field descriptions

Field	Description
MS[95:64]	*Multiple input Signature, bits 95-64* These bits represent the MISR value obtained 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 DFLASH_UMISR2.

DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3)

The DFLASH_UMISR3 provides a mean to evaluate the Array Integrity.

The DFLASH_UMISR3 represents the bits 127:96 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH_UMISR3 is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 381. DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3)

Address offset: 0x00054 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS[127:112]															
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[111:96]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 367. DFLASH_UMISR3 field descriptions

Field	Description
MS[127:96]	*Multiple input Signature, bits 127096* These bits represent the MISR value obtained accumulating the bits 127:96 of all the pages read from the flash memory. The MS can be seeded to any value by writing the DFLASH_UMISR3.

DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity.

The User Multiple Input Signature Register 4 represents the ECC bits of the whole 144 bits word (2 Double Words including ECC): bits 23-168:15 are ECC bits for the odd Double Word and bits 7-024:31 are the ECC bits for the even Double Word; bits 27-264:5 and 11-1020:21 of MISR are respectively the double and single ECC error detection for odd and even Double Word.

The DFLASH_UMISR4 Register is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Figure 382. DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4)

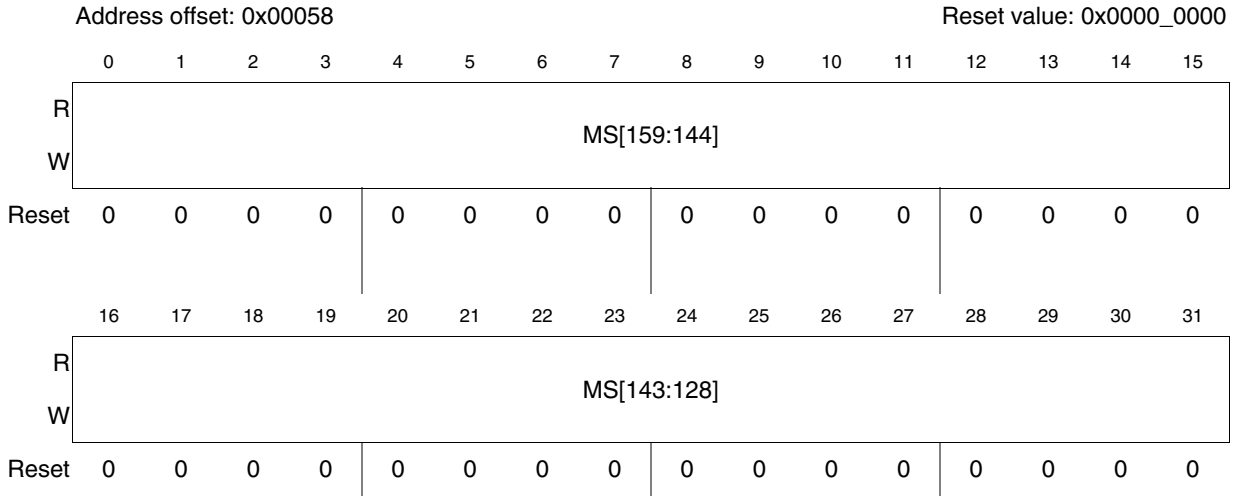


Table 368. DFLASH_UMISR4 field descriptions

Field	Description
MS[159:128]	*Multiple input Signature, bits 159-128* These bits represent the MISR value obtained accumulating: the 8 ECC bits for the even Double Word (on MS[135:128]); the single ECC error detection for even Double Word (on MS138); the double ECC error detection for even Double Word (on MS139); the 8 ECC bits for the odd Double Word (on MS[151:144]); the single ECC error detection for odd Double Word (on MS154); the double ECC error detection for odd Double Word (on MS155). The MS can be seeded to any value by writing the DFLASH_UMISR4 register.

28.6 Programming considerations

In the following sections, register names can refer to the CFlash or DFlash versions of those registers. Thus, for example, the term “MCR” can refer to the CFLASH_MCR or DFLASH_MCR based on context.

28.6.1 Modify operation

All modify operations of the flash memory module are managed through the flash memory User Registers Interface.

All the sectors of the flash memory 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-Write is not supported).

During a flash memory modify operation any attempt to read any flash memory location will output invalid data and bit MCR[RWE] will be automatically set. This means that the flash memory module is not fetchable when a modify operation is active and these commands must be executed from another memory (internal SRAM or another flash memory module).

If during a Modify Operation a reset occurs, the operation is suddenly terminated and the Macrocell is reset to Read Mode. The data integrity of the flash memory section where the

Modify Operation has been terminated is not guaranteed: the interrupted flash memory Modify Operation must be repeated.

In general each modify operation is started through a sequence of three steps:

1. The first instruction is used to select the desired operation by setting its corresponding selection bit in MCR (PGM or ERS) or UT0 (MRE or EIE).
2. The second step is the definition of the operands: the Address and the Data for programming or the Sectors for erase or margin read.
3. The third instruction is used to start 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 memory modify operations is shown in *Table 369*.

Table 369. Flash memory 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 the MCR[EHV] bit (or UT0[AIE]) is set, all the operands can no more be modified until the MCR[DONE] bit (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 the MCR[DONE] bit (or UT0[AID]) to go high.
2. Check operation result: check the MCR[PEG] bit (or compare UMISR0-4 with expected value).
3. Switch off FPEC by resetting the MCR[EHV] bit (or UT0[AIE]).
4. Deselect current operation by clearing the MCR[PGM] / MCR[ERS] fields (or UT0[MRE] /UT0[EIE]).

If the device embeds more than one flash memory module and a modify operation is on-going on one of them, then it is forbidden to start any other modify operation on the other flash memory modules.

In the following all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

28.6.2 Double word program

A flash memory Program sequence operates on any Double Word within the flash memory core.

Up to two words within the Double Word may be altered in a single Program operation.

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 two words, of a Double Word, with a single program sequence.

Double Word-bound words have addresses which 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.
   Write the first address to be programmed with the program data.
   The flash memory module latches address bits (22:3) at this time.
   The flash memory module latches data written as well.
   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 memory module ignores address bits (22:3) for program data writes.
   The eventual unwritten data word default to 0xFFFFFFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm that the MCR[PEG] bit is 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.

Program 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 if the shadow, test or normal array space will be programmed by causing the MCR[PEAS] field 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 0xFFFFFFFF. 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 EHV, resulting in a program abort.
A Program abort forces the module to step 8 of the program sequence.

An aborted program will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The data space being operated on before the abort will contain indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

Example 5 Double word program of data 0x55AA55AA at address 0x00AAA8 and data 0xAA55AA55 at address 0x00AAAC

```
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 */
```

28.6.3 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 sector (if available). The test block cannot be erased.

The erase sequence is fully automated within the flash memory. 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 ‘1’s to the appropriate bit(s) in the LMS register.
   If the shadow sector is to be erased, this step may be skipped, and LMS is ignored.
   Note that Lock and Select are independent. If a block is selected and locked, no erase will occur.
3. Write to any address in flash memory. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit 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. Write a logic 0 to 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 ERS before setting EHV.

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high and MCR[ESUS] is low.

An erase abort forces the module to step 8 of the erase sequence.

An aborted erase will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The block(s) being operated on before the abort contain indeterminate data. This may be recovered by executing an erase on the affected blocks.

The user may not abort an erase sequence while in erase suspend.

Example 6 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 memory 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 */
```

Erase suspend/resume

The erase sequence may be suspended to allow read access to the flash memory 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 which 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 memory core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

Example 7 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 8 Sector erase resume

```
MCR        = 0x00000005;        /* Reset ESUS in MCR: Erase Resume */
```

User Test mode

The user can perform specific tests to check flash memory module integrity by putting the flash memory module in User Test Mode.

Three kinds of test can be performed:

- Array Integrity Self Check
- Margin Read
- ECC Logic Check

The User Test Mode is equivalent to a Modify operation: read accesses attempted by the user during User Test Mode generates a Read-While-Write Error (MCR[RWE] set).

It is not allowed to perform User Test operations on the Test and shadow sectors.

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 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 will scan only bits 31:0 of each page.
2. The second pass will scan only bits 63:32 of each page.
3. The third pass will scan only bits 95:64 of each page.
4. The fourth pass will scan only bits 127:96 of each page.
5. The fifth pass will scan 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 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 ‘1’s to the appropriate bit(s) in 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 goes high.
6. Compare UMISR0-4 content with the expected result.
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. 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 AIE, resulting in a Array Integrity Check abort.

UT0[AID] must be checked to know when the aborting command has completed.

Example 9 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

Margin read procedure (either Margin 0 or Margin 1), can be run on unlocked blocks in order to unbalance the Sense Amplifiers, 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 comparing checksum value in UMISR0-4. Since Margin reads are done at voltages that differ than the normal read voltage, lifetime expectancy of the flash memory macrocell is impacted by the execution of Margin reads. Doing Margin reads repetitively results in degradation of the flash memory Array, and shorten expected lifetime experienced at normal read levels. For these reasons the Margin Read usage is allowed only in Factory, while it is forbidden to use it inside the User Application.

In any case the charge losses detected through the Margin Read 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 UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1’s to the appropriate bit(s) in 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 T0.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 0's margin, UT0[MRV]=1 for 1's margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read Setup or skip to step 6 to terminate.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR0-4 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, the usage of the proprietary sequence in Margin Read is forbidden.

During the execution of the Margin Read 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 abort.

UT0[AID] must be checked to know when the aborting command has completed.

Example 10 Margin read setup versus '1's

```
UMISR0     = 0x00000000;   /* Reset UMISR0 content */
UMISR1     = 0x00000000;   /* Reset UMISR1 content */
UMISR2     = 0x00000000;   /* Reset UMISR2 content */
UMISR3     = 0x00000000;   /* Reset UMISR3 content */
UMISR4     = 0x00000000;   /* Reset UMISR4 content */
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. */
```

To exit from the Margin Read Mode a Read Reset operation must be executed.

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 in UT1[DAI31–0] and UT2[DAI63–32] the Double Word Input value.
3. Write in UT0[DSI7–0] the Syndrome Input value.
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 UMISR0–4 content with the expected result.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] is low UMISR0–4, UT1–2 and bits MRE, MRV, EIE, AIS and DSI7–0 of UT0 are not accessible: reading returns indeterminate data and write has no effect.

Example 11 ECC 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

The flash memory module provides a method to improve the reliability of the data stored in flash memory: the usage of an Error Correction Code. The word size is fixed at 64 bits.

Eight ECC bits, programmed to guarantee a Single Error Correction and a Double Error Detection (SEC-DED), are associated to each 64-bit Double Word.

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 to product reliability.

ECC algorithms

The flash memory module supports one ECC Algorithm: “All ‘1’s No Error”. A modified Hamming code is used that ensures the all erased state (that is, 0xFFFF.....FFFF) data is a

valid state, and will not cause an ECC error. This allows the user to perform a blank check after a sector erase operation.

EEPROM emulation

The choosen 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 choosen ECC algorithm allows to start from an All ‘1’s Double Word value and rewrite whichever of its four 16-bits Half-Words to an All ‘0’s content by keeping the same ECC value.

Table 370 shows a set of Double Words sharing the same ECC value.

Table 370. Bit manipulation: Double words with the same ECC value

Double word	ECC all ‘1’s 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 memory sectors are used to perform an Eeprom Emulation, it is reccomended for safety reasons to reserve at least 3 sectors to this purpose.

All ‘1’s No Error

The All ‘1’s No Error Algorithm detects as valid any Double Word read on a just erased sector (all the 72 bits are ‘1’s).

This option allows to perform a Blank Check after a Sector Erase operation.

Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in flash memory sectors and Censored Mode to avoid piracy.

Modify protection

The flash memory Modify Protection information is stored in nonvolatile flash memory cells located in the TestFlash. This information is read once during the flash memory initialization phase following the exiting from Reset and is stored in volatile registers that act as actuators.

The reset state of all the volatile modify protection registers is the protected state.

All the nonvolatile modify protection registers can be programmed through a normal Double Word Program operation at the related locations in TestFlash.

The nonvolatile modify protection registers cannot be erased.

- The nonvolatile Modify Protection Registers are physically located in TestFlash their bits can be programmed to ‘0’ only once and they can no more be restored to ‘1’.
- The Volatile Modify Protection Registers are Read/Write registers which bits 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 register.

An alternate means to enable software locking for blocks of Low Address Space only is through the SLL.

All these registers have a nonvolatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash nonvolatile image is at all ‘1’s, meaning all sectors are locked.

By programming the nonvolatile locations in TestFlash the selected sectors can be unlocked.

Being the TestFlash One Time Programmable (that is, not erasable), once unlocked the sectors 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 nonvolatile flash memory cells located in the Shadow Sector. This information is read once during the flash memory initialization phase following the exiting from Reset and is stored in volatile registers that act as actuators.

The reset state of all the Volatile Censored Mode Registers is the protected state.

All the nonvolatile Censored Mode registers can be programmed through a normal Double Word Program operation at the related locations in the Shadow Sector.

The nonvolatile Censored Mode registers can be erased by erasing the Shadow Sector.

- The nonvolatile Censored Mode Registers are physically located in the Shadow Sector their bits can be programmed to ‘0’ and restored to ‘1’ by erasing the Shadow Sector.
- The Volatile Censored Mode Registers are registers not accessible by the user application.

The flash memory module provides two levels of protection against piracy:

- If bits CW15:0 of NVSCC0 are programmed at 0x55AA and NVSC1 = NVSCC0 the Censored Mode is disabled, while all the other possible values enable the Censored Mode.
- If bits SC15:0 of NVSCC0 are programmed at 0x55AA and NVSC1 = NVSCC0 the Public Access is disabled, while all the other possible values enable the Public Access.

The parts are delivered to the user with Censored Mode and Public Access disabled.

28.7 Platform flash memory controller

28.7.1 Introduction

The platform flash memory controller acts as the interface between the system bus (AHB-Lite 2.v6) and up to two banks of integrated flash memory arrays (Program and Data). It intelligently converts the protocols between the system bus and the dedicated flash memory array interfaces.

A block diagram of the e200z0h Power Architecture reduced product platform (RPP) reference design is shown below in *Figure 383* with the platform flash memory controller module and its attached off-platform flash memory arrays highlighted.

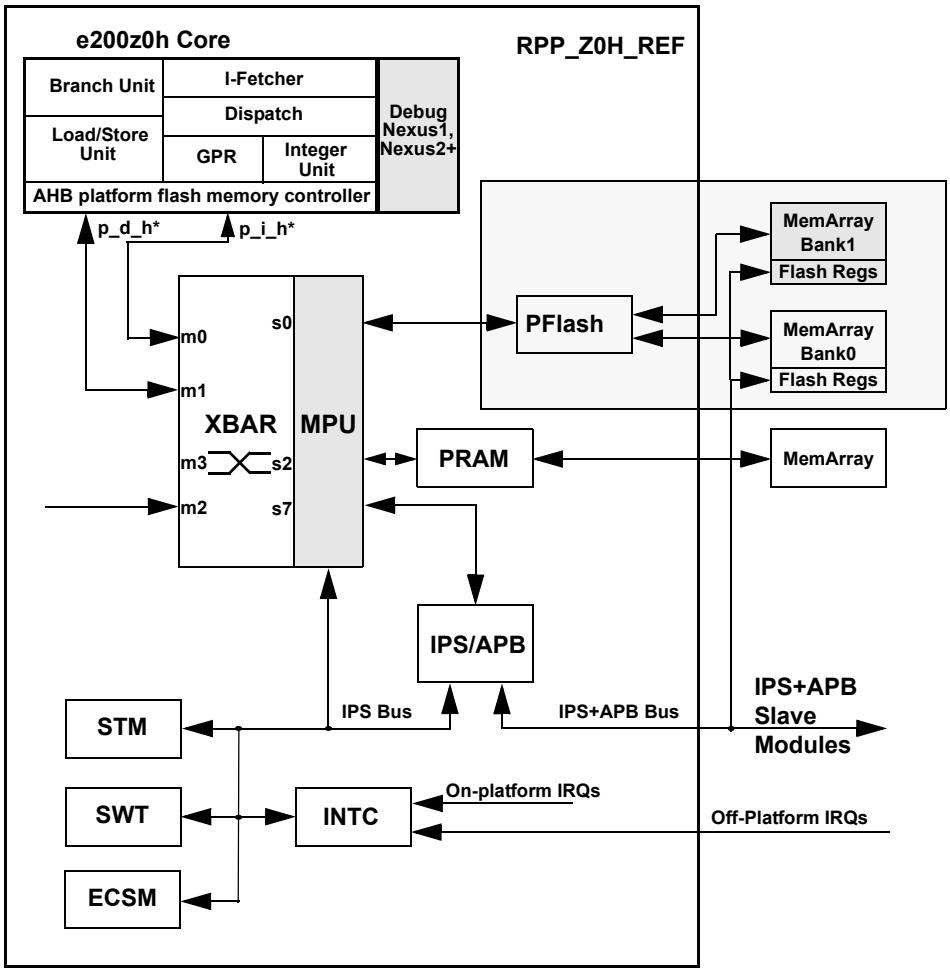


Figure 383. Power Architecture e200z0h RPP reference platform block diagram

The module list includes:

- Power Architecture e200z0h (Harvard) core with Nexus1 or optional Nexus2+ debug
- AHB crossbar switch “lite” (XBAR)
- Memory Protection Unit (MPU)
- Platform flash memory controller with connections to 2 memory banks
- Platform SRAM memory controller (PRAM)
- AHB-to-IPS/APB bus controller (PBRIDGE) for access to on- and off-platform slave modules
- Interrupt Controller (INTC)
- 4-channel System Timers (STM)
- Software Watchdog Timer (SWT)
- Error Correction Status Module (ECSM)

Throughout this document, several important terms are used to describe the platform flash memory controller module and its connections. These terms are defined here:

- **Port** — This is used to describe the AMBA-AHB connection(s) into the platform flash memory controller. From an architectural and programming model viewpoint, the definition supports up to two AHB ports, even though this specific controller only supports a single AHB connection.
- **Bank** — This term is used to describe the attached flash memories. From the platform flash memory controller's perspective, there may be one or two attached banks of flash memory. The "code flash memory" is required and always attached to bank0. Additionally, there is a "data flash memory" attached to bank1. The platform flash memory controller interface supports two separate connections, one to each memory bank.
- **Array** — Within each memory bank, there is one flash memory array instantiations.
- **Page** — This value defines the number of bits read from the flash memory 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 memory 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 instantiations of the flash memory array. One flash memory bank is connected to the code flash memory and the other bank is connected to the optional 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 memory controller logic associated with the code flash memory bank contains a four-entry "page" buffer, each entry containing 128 bits of data (1 flash memory page) plus an associated controller which prefetches sequential lines of data from the flash memory array into the buffer, while the controller logic associated with the data flash memory bank only supports a 128-bit register which serves as a temporary page holding register and does not support any prefetching. Prefetch buffer hits from the code flash memory bank support zero-wait AHB data phase responses. AHB read requests which miss the buffers generate the needed flash memory array access and are forwarded to the AHB upon completion, typically incurring two wait-states at an operating frequency of 60–64 MHz.

This memory controller is optimized for applications where a cacheless processor core, e.g., the Power e200z0h, is connected through the platform to on-chip memories, e.g., flash memory and SRAM, 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 zero 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 memory controller:

- Dual array interfaces support up to a total of 16 MB of flash memory, partitioned as two separate 8 MB banks
- 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 memory (bank0) provides configurable read buffering and page prefetch support. Four page read buffers (each 128 bits wide) and a prefetch controller are used to support single-cycle read responses (zero AHB data phase wait-states) for hits in the buffers. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- Interface with optional data flash memory (bank1) includes a 128-bit register to temporarily hold a single flash memory page. This logic supports single-cycle read responses (zero AHB data phase wait-states) for accesses that hit in the holding register. There is no support for prefetching associated with this bank.
- Programmable response for read-while-write sequences including support for stall-while-write, optional stall notification interrupt, optional flash memory operation abort, and optional abort 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 memory ECC events
- Typical operating configuration loaded into programming model by system reset

28.7.2 Memory map and register description

Two memory maps are associated with the platform flash memory 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 and 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 memory controller. Rather, the platform flash memory controller receives control and configuration information from the flash memory array controller(s) to determine the operating configuration. These are part of the flash memory array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

Memory map

First, consider the flash memory space accessed via transactions from the platform flash memory controller's AHB port.

To support the two separate flash memory banks, each up to 8 MB in size, the platform flash memory controller uses address bit 23 (haddr[23]) to steer the access to the appropriate memory bank. In addition to the actual flash memory regions, the system memory map includes shadow and test sectors. 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 memory array and one data flash memory array. See *Table 371*.

Table 371. Flash memory-related regions in the system memory map

Start address	End address	Size [KB]	Region
0x0000_0000	0x0007_FFFF	512	Code flash memory array 0
0x0008_0000	0x001F_FFFF	1536	Reserved
0x0020_0000	0x0027_FFFF	16	Code flash memory array 0: shadow sector
0x0028_0000	0x002F_FFFF	1536	Reserved
0x0040_0000	0x0040_3FFF	16	Code flash memory array 0: test sector
0x0040_4000	0x007F_FFFF	4078	Reserved
0x0080_0000	0x0080_FFFF	64	Data flash memory array 0
0x0081_0000	0x00BF_FFFF	4032	Reserved
0x00C0_0000	0x00C7_FFFF	16	Data flash memory array 0: test sector
0x00C8_0000	0x00FF_FFFF	3584	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Emulation mapping
0xC3F8_8000	0xC3F8_BFFF	16	Code flash memory array 0 configuration
0xC3F8_C000	0xC3F8_FFFF	16	Data flash memory array 0 configuration

For additional information on the address-based read access timing for emulation of other memory types, see *Section 28.8.11, Wait-state emulation*.

Next, consider the memory map associated with the control and configuration registers.

Regardless of the number of populated banks or the number of flash memory arrays included in a given bank, the configuration of the platform flash memory controller is wholly specified by the platform flash memory controller registers associated with code flash memory array 0. The code array0 register settings define the operating behavior of **both** flash memory banks; it is recommended that the platform flash memory controller registers for all physically-present arrays be set to the code flash memory array0 values.

Note: To perform program and erase operations, the control registers in the actual referenced flash memory array must be programmed, but the configuration of the platform flash memory controller module is defined by the platform flash controller registers of code array0.

The 32-bit memory map for the platform flash memory controller control registers is shown in *Table 372*. The base address of the controller is 0xC3F8_8000.

Table 372. Platform flash memory controller 32-bit memory map

Address offset	Register	Location
0x1C	Platform Flash Configuration Register 0 (PFCR0)	*on page 28-708*
0x20	Platform Flash Configuration Register 1 (PFCR1)	*on page 28-711*
0x24	Platform Flash Access Protection Register (PFAPR)	*on page 28-713*

See the SPC560Bx and SPC560Cx data sheet for detailed settings for different values of frequency.

Register description

This section details the individual registers of the platform flash memory controller.

Flash memory configuration registers must be written only with 32-bit write operations to avoid any issues associated with register “incoherency” caused by bits spanning smaller-size (8- or 16-bit) boundaries.

Platform Flash Configuration Register 0 (PFCR0)

This register defines the configuration associated with the code flash memory bank0. It includes fields that provide specific information for up to two separate AHB ports (p0 and the optional p1). For the platform flash memory controller module, the fields associated with AHB port p1 are ignored. The register is described in *Figure 384* and *Table 373*.

Note: Do not execute code from flash memory when you are programming PFCR0. If you wish to program PFCR0, execute your application code from RAM.

Figure 384. PFlash Configuration Register 0 (PFCR0)

Offset 0x01C Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BK0_APC				BK0_WWSC				BK0_RWSC							BK0_RWWC
W																
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	BK0_RWWC	0	0	0	0	0	0	0	BK0_RWWC	B0_P0_BCFG		B0_P0_DPFE	B0_P0_IPFE	B0_P0_PFLM		B0_P0_BFE
W																
Reset	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1

Table 373. PFCR0 field descriptions

Field	Description
BK0_APC	Bank0 Address Pipelining Control This field is used to control the number of cycles between flash memory array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. 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 *Note:*
BK0_WWSC	Bank0 Write Wait-State Control This field is used to control the number of wait-states to be added to the flash memory array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. This field is set to an appropriate value by hardware reset. 00000: No additional wait-states are added 00001: One additional wait-state is added 00010: Two additional wait-states are added ... 11111: 31 additional wait-states are added *Note:*
BK0_RWSC	Bank0 Read Wait-State Control This field is used to control the number of wait-states to be added to the flash memory 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. The required settings are documented in the device datasheet. 00000: No additional wait-states are added 00001: One additional wait-state is added 00010: Two additional wait-states are added ... 11111: 31 additional wait-states are added

Table 373. PFCR0 field descriptions (continued)

Field	Description
BK0_RWWC	Bank0 Read-While-Write Control This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation. 0—: This state should be avoided. Setting to this state can cause unpredictable operation. 111: Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + 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 101: Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt 100: Generate a bus stall for a read while write/erase, enable the operation abort and the 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.
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 memory 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 memory 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.
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. Prefetching can be enabled/disabled on a per Master basis at PFAPR[MxPFD]. 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

Table 373. PFCR0 field descriptions (continued)

Field	Description
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. Prefetching can be enabled/disabled on a per Master basis at PFAPR[MxPFD]. 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
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–: 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*.
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)

This register defines the configuration associated with flash memory bank1. This corresponds to the “data flash memory”. It includes fields that provide specific information for up to two separate AHB ports (p0 and the optional p1). For the platform flash memory controller module, the fields associated with AHB port p1 are ignored. The register is described below in *Figure 385* and *Table 374*.

Note: Do not execute code from flash memory when you are programming PFCR1. If you wish to program PFCR1, execute your application code from RAM.

Figure 385. PFlash Configuration Register 1 (PFCR1)

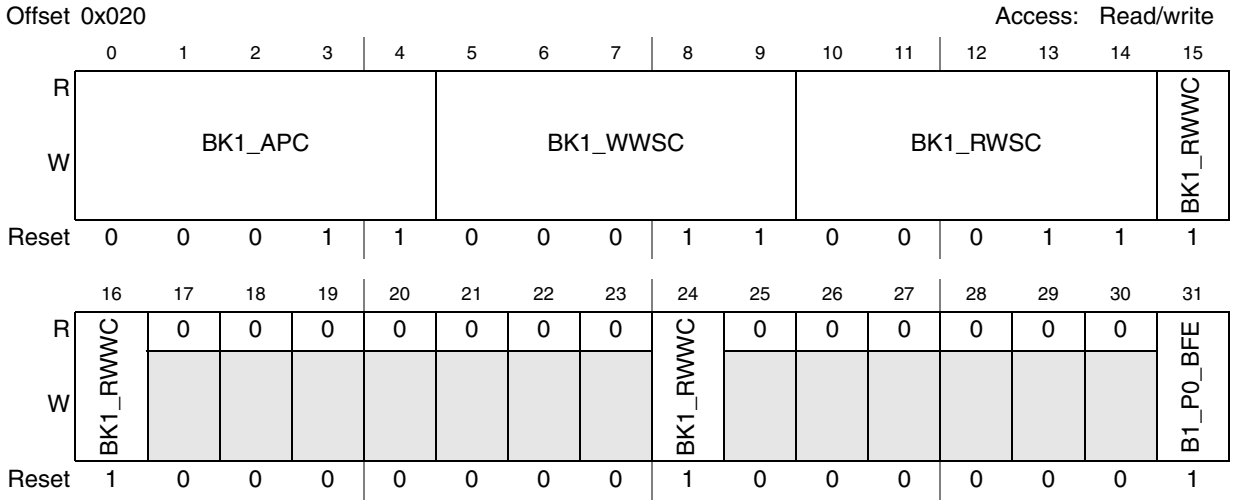


Table 374. PFCR1 field descriptions

Field	Description
BK1_APC	Bank1 Address Pipelining Control This field is used to control the number of cycles between flash memory array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. 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 This field is ignored in single bank flash memory configurations. *Note:*
BK1_WWSC	Bank1 Write Wait-State Control This field is used to control the number of wait-states to be added to the flash memory array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. This field is set to an appropriate value by hardware reset. 00000: No additional wait-states are added 00001: One additional wait-state is added 00010: Two additional wait-states are added ... 11111: 31 additional wait-states are added This field is ignored in single bank flash memory configurations.

Table 374. PFCR1 field descriptions (continued)

Field	Description
BK1_RWSC	Bank1 Read Wait-State Control This field is used to control the number of wait-states to be added to the flash memory 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. The required settings are documented in the device datasheet. 00000: No additional wait-states are added 00001: One additional wait-state is added 00010: Two additional wait-states are added ... 11111: 31 additional wait-states are added This field is ignored in single bank flash memory configurations.
BK1_RWWC	Bank1 Read-While-Write Control This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation. 0—: Terminate any attempted read while write/erase with an error response 111: Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + 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 101: Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt 100: Generate a bus stall for a read while write/erase, enable the operation abort and the 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. This field is ignored in single bank flash memory configurations.
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 PFlash Access Protection Register (PFAPR) is used to control read and write accesses to the flash memory 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 below in *Figure 386* and *Table 375*.

The contents of the register are loaded from location 0x203E00 of the shadow region in the code flash memory (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 0x203E00 of the shadow region in the flash memory array must be programmed using the normal sequence of operations. The reset value shown in *Table 386* reflects an erased or unprogrammed value from the shadow region.

Figure 386. PFlash Access Protection Register (PFAPR)

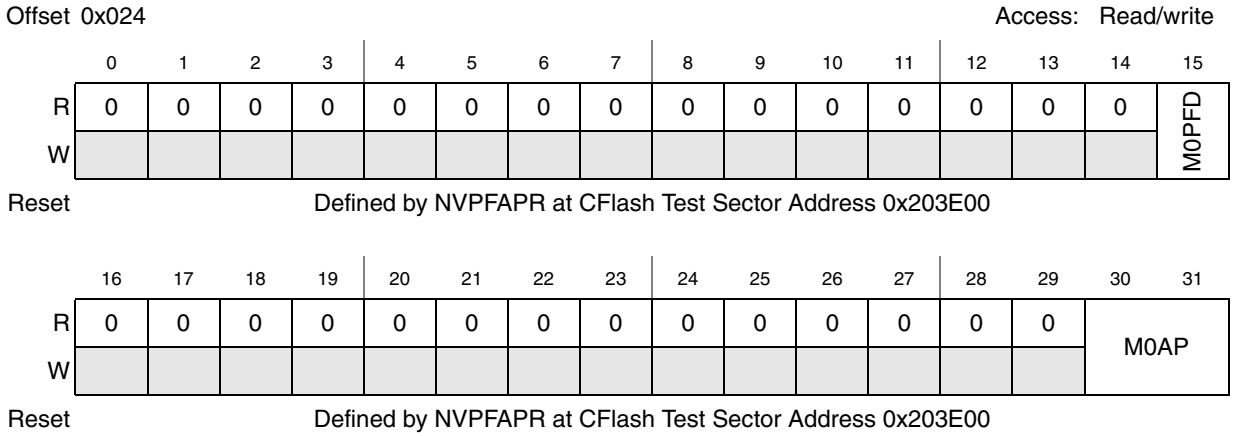


Table 375. PFAPR field descriptions

Field	Description
M0PFD	e200z0 core Master 0 Prefetch Disable This field controls whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_Px_DPFE, B0_Px_IPFE, Bx_Py_BFE] bits. For master numbering, see *Table 125*. 0: Prefetching may be triggered by this master 1: No prefetching may be triggered by this master
M0AP	e200z0 core Master 0 Access Protection These fields control whether read and write accesses to the flash memory are allowed based on the master number of the initiating module. For master numbering, see *Table 125*. 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

Nonvolatile Platform Flash Access Protection Register (NVPFAPR)

The NVPFAPR register has a related Nonvolatile PFAPR located in the Shadow Sector that contains the default reset value for PFAPR. During the reset phase of the flash memory module, the NVPFAPR register content is read and loaded into the PFAPR.

The NVPFAPR register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Figure 387. Nonvolatile Platform Flash Access Protection Register (NVPFAPR)

Offset: 0x203E00 Access: Read/write

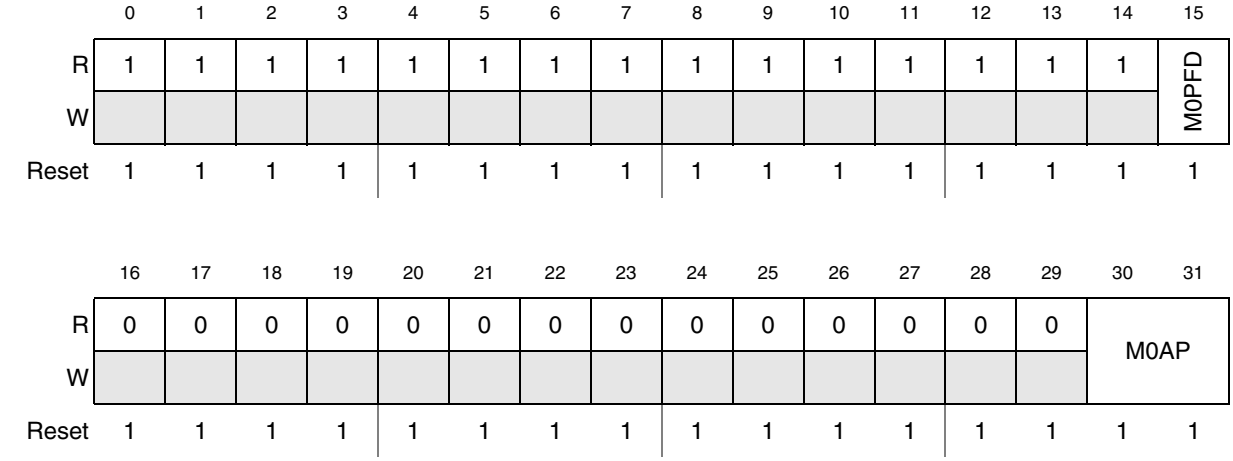


Table 376. NVPFAPR field descriptions

Field	Description
M0PFD	See *Table 375*.
M0AP	See *Table 375*.

28.8 Functional description

The platform flash memory controller interfaces between the AHB system bus and the flash memory arrays.

The platform flash memory controller generates read and write enables, the flash memory array address, write size, and write data as inputs to the flash memory array. The platform flash memory controller captures read data from the flash memory array interface and drives it onto the AHB. Up to four pages of data (128-bit width) from bank0 are buffered by the platform flash memory controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (zero 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; see *Section , Platform Flash Configuration Register 0 (PFCR0)*, and *Section , Platform Flash Configuration Register 1 (PFCR1)*.

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms; see *Section , Platform Flash Access Protection Register (PFAPR)*.

Throughout this discussion, bkn_ is used as a prefix to refer to two signals, each for each bank: bk0_ and bk1_. Also, the nomenclature Bx_Py_RegName is used to reference a program-visible register field associated with bank “x” and port “y”.

28.8.1 Access protections

The platform flash memory controller provides programmable configurable access protections for both read and write cycles from masters via the PFlash 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 memory controller on the AHB transfer.

28.8.2 Read cycles – Buffer miss

Read cycles from the flash memory array are initiated by the platform flash memory controller. The platform flash memory controller then waits for the programmed number of read wait-states before sampling the read data from the flash memory array. 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.

If the flash memory 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 memory 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.

28.8.3 Read cycles – Buffer hit

Single cycle read responses to the AHB are possible with the platform flash memory 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 zero wait-state response.

Likewise, the bank1 logic includes a single 128-bit temporary holding register and sequential accesses which “hit” in this register are also serviced with a zero wait-state response.

28.8.4 Write cycles

Write cycles are initiated by the platform flash memory controller. The platform flash memory controller then waits for the appropriate number of write wait-states before terminating the write operation.

28.8.5 Error termination

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 memory controller does not initiate a flash memory array access.

The second case that can cause an error response to the AHB is when an access is performed to the flash memory array and is terminated with a flash memory error response. See *Section 28.8.7, Flash error response operation*. This may occur for either a read or a write operation.

A third case involves an attempted read access while the flash memory 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.

28.8.6 Access pipelining

The platform flash memory controller does not support access pipelining since this capability is not supported by the flash memory 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, BKn_APC = BKn_RWSC. It cannot be less than the RWSC.

28.8.7 Flash error response operation

The flash memory array may signal an error response 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 memory 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 additional information on the system registers which capture the faulting address, attributes, data and ECC information, see the chapter “Error Correction Status Module (ECSM).”

28.8.8 Bank0 page read buffers and prefetch operation

The logic associated with bank0 of the platform flash memory controller contains four 128-bit page read buffers which are used to hold instructions and data read from the flash memory array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

For the general case, a page buffer is written at the completion of an error-free flash memory access and the valid bit asserted. Subsequent flash memory 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 28.8.7, Flash error response operation*, a page buffer is *not* marked as valid if the flash memory array access terminated with any type of transfer error. However, the result is that flash memory 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 memory 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 memory 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 order of priority:

1. Invalid — The buffer contains no valid data.
2. Used — The buffer contains valid data which has been provided to satisfy an AHB burst type read.
3. Valid — The buffer contains valid data which has been provided to satisfy an AHB single type read.
4. Prefetched — The buffer contains valid data which 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 memory 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 memory 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 memory accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash memory access.

Several algorithms are available for prefetch control which trade off performance versus power. They are defined by the Bx_Py_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 (PFCRn[Bx_Py_BFE]) must be set, the prefetch limit (PFCRn[Bx_Py_PFLM]) must be non-zero, either instruction prefetching (PFCRn[Bx_Py_IPFE]) or data prefetching (PFCRn[Bx_Py_DPFE]) enabled, and Master Access must be enabled (PFAPR[MxPFD]). See *Section , Register description*, for a description of these control fields.

Instruction/Data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the Bx_Py_IPFE control field, while prefetching for data reads is enabled via the Bx_Py_DPFE control field. Additionally, the Bx_Py_PFLIM field must 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. See *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 (Bx_Py_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.

At the beginning of all program/erase operations, the flash memory array will invalidate the page read buffers. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer which is in progress.

Software may invalidate the buffers by clearing the Bx_Py_BFE bit, which also disables the buffers. Software may then re-assert the Bx_Py_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 memory data which 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 memory 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.

28.8.9 Bank1 Temporary Holding Register

Recall the bank1 logic within the platform flash memory controller 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_P0_BFE).

For the general case, a temporary holding register is written at the completion of an error-free flash memory access and the valid bit asserted. Subsequent flash memory 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 flash memory array at the beginning of all program/erase operations 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_P0_BFE register bit can be cleared by software to invalidate the contents of the holding register.

As noted in *Section 28.8.7, Flash error response operation*, the temporary holding register is *not* marked as valid if the flash memory array access terminated with any type of transfer

error. However, the result is that flash memory 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 memory data which 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.

28.8.10 Read-while-write functionality

The platform flash memory controller supports various programmable responses for read accesses while the flash memory is busy performing a write (program) or erase operation. For all situations, the platform flash memory controller uses the state of the flash memory array’s MCR[DONE] output to determine if it is busy performing some type of high voltage operation, namely, if MCR[DONE] = 0, the array is busy.

Specifically, two 3-bit read-while-write (BKn_RWWC) control register fields define the platform flash memory controller’s response to these types of access sequences. Five unique responses are defined by the BKn_RWWC setting: one that immediately reports an error on an attempted read and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- BKn_RWWC = 0b0--

  For this mode, any attempted flash memory 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 memory array.

- BKn_RWWC = 0b111

  This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform flash memory controller module simply stalls any read reference until the flash memory has completed its program/erase operation. If a read access arrives while the array is busy or if MCR[DONE] goes low while a read is still in progress, the AHB data phase is stalled and the read access address is saved. Once the array has completed its program/erase operation, the platform flash memory 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. 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 to terminate the system bus transfer.

- BKn_RWWC = 0b110

  This setting is similar to the basic stall-while-write capability provided when BKn_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 (see the INTC chapter of this reference manual).

- BKn_RWWC = 0b101

  Again, this setting provides the basic stall-while-write capability with the added ability to abort 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 aborted by the platform flash memory controller. For this setting, no notification interrupts are generated.

- BKn_RWWC = 0b100

  This setting provides the basic stall-while-write capability with the ability to abort any program/erase operation if a read access is initiated plus the generation of an abort 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 aborted by the platform flash memory controller and an abort notification interrupt generated. There are two abort notification interrupts, one for each bank.

As detailed above, a total of four interrupt requests are 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 377. Platform flash memory controller stall-while-write interrupts

MIR[n]	Interrupt description
ECSM.MIR[0]	Platform flash memory bank0 abort notification, MIR[FB0AI]
ECSM.MIR[1]	Platform flash memory bank0 stall notification, MIR[FB0SI]
ECSM.MIR[2]	Platform flash memory bank1 abort notification, MIR[FB1AI]
ECSM.MIR[3]	Platform flash memory bank1 stall notification, MIR[FB1S1]

28.8.11 Wait-state emulation

Emulation of other memory array timings are supported by the platform flash memory controller on read cycles to the flash memory. This functionality may be useful to maintain the access timing for blocks of memory which were used to overlay flash memory blocks for the purpose of system calibration or tuning during code development.

The platform flash memory controller inserts additional wait-states according to the values of haddr[28:24]. 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 378 and *Table 379* 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 memory read by (8 * haddr[25:24] + haddr[28:26]) cycles.

Table 378. Additional wait-state encoding

Memory address haddr[28:26]	Additional wait-states
000	0
001	1
010	2
011	3

Table 378. Additional wait-state encoding

Memory address haddr[28:26]	Additional wait-states
100	4
101	5
110	6
111	7

Table 379 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 379. 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

29 Register Protection

29.1 Introduction

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 protection module is located between the module under protection and the peripheral bridge. This is shown in *Figure 388*.

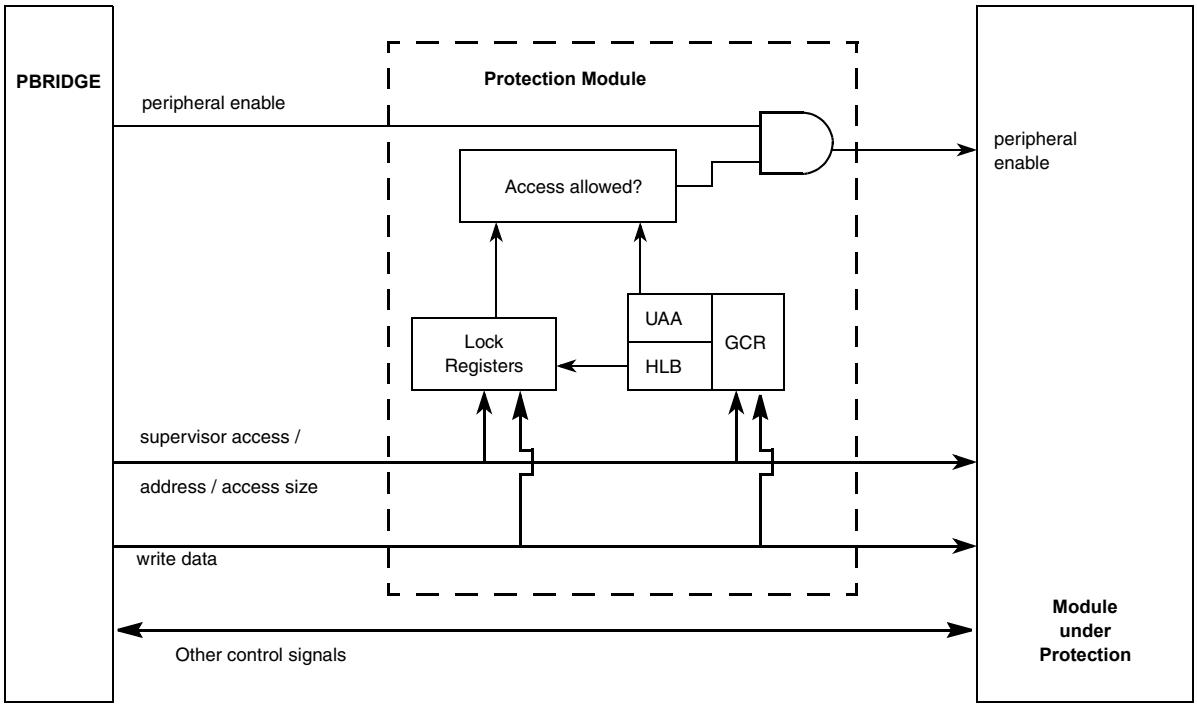


Figure 388. Register Protection block diagram

Please see the “Registers Under Protection” appendix for the list of protected registers.

29.2 Features

The Register Protection includes these distinctive 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

29.3 Modes of operation

The Register Protection module is operable when the module under protection is operable.

29.4 External signal description

There are no external signals.

29.5 Memory map and register 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 389*.

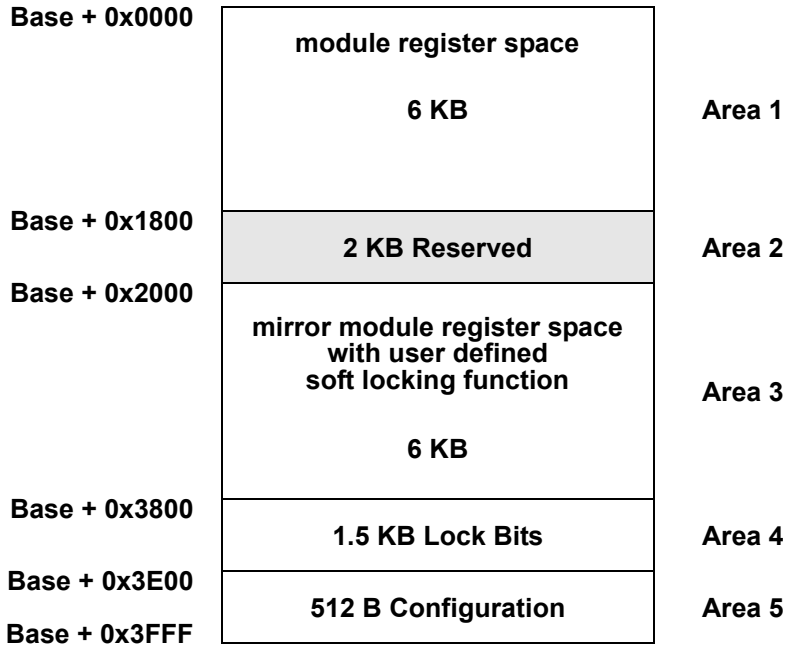


Figure 389. Register protection memory diagram

Area 1 spans 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2 spans 2 KB starting at address 0x1800. It is a reserved area, which cannot be accessed.

Area 3 spans 6 KB, starting at address 0x2000 and is a mirror of area 1. A read/write access to a 0x2000+X address will reads/writes the register at address X. As a side effect, a write access to address 0x2000+X sets the optional soft lock bits for 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 are 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 a 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 byte 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 results in a transfer error.

29.5.1 Memory map

Table 380 gives an overview on the Register Protection registers implemented.

Table 380. Register protection memory map

Address offset	Register	Location
0x0000	Module Register 0 (MR0)	*on page 29-726*
0x0001	Module Register 1 (MR1)	*on page 29-726*
0x0002	Module Register 2 (MR2)	*on page 29-726*
0x0003–0x17FF	Module Register 3 (MR3) - Module Register 6143 (MR6143)	*on page 29-726*
0x1800–0x1FFF	Reserved	—
0x2000	Module Register 0 (MR0) + Set soft lock bit 0 (LMR0)	*on page 29-726*
0x2001	Module Register 1 (MR1) + Set soft lock bit 1 (LMR1)	*on page 29-726*
0x2002–0x37FF	Module Register 2 (MR2) + Set soft lock bit 2 (LMR2) – Module Register 6143 (MR6143) + Set soft lock bit 6143 (LMR6143)	*on page 29-726*
0x3800	Soft Lock Bit Register 0 (SLBR0): soft lock bits 0-3	*on page 29-726*
0x3801	Soft Lock Bit Register 1 (SLBR1): soft lock bits 4-7	*on page 29-726*
0x3802–0x3DFF	Soft Lock Bit Register 2 (SLBR2): soft lock bits 8-11 – Soft Lock Bit Register 1535 (SLBR1535): soft lock bits 6140-6143	*on page 29-726*
0x3E00–0x3FFB	Reserved	—
0x3FFC	Global Configuration Register (GCR)	*on page 29-727*

Note: Reserved registers in area #2 will be handled according to the protected IP (module under protection).

29.5.2 Register description

Module Registers (MR0-6143)

This is the lower 6 KB module memory space which 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 381*.

Soft Lock Bit Register (SLBR0-1535)

These registers hold the soft lock bits for the protected registers in memory area #1.

Figure 390. Soft Lock Bit Register (SLBR*n*)

Address 0x3800-0x3DFF

Access: Read always
Supervisor 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 381. 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 1 Value is written to SLB 0 SLB is not modified
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] 1 Associated MRn byte is locked against write accesses 0 Associated MRn byte is unprotected and writeable

Figure 382 gives some examples how SLBR*n*.SLB and MR*n* go together.

Table 382. 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)

This register is used to make global configurations related to register protection.

Figure 391. Global Configuration Register (GCR)

Address 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 383. GCR field descriptions

Field	Description
HLB	Hard Lock Bit. This register can not be cleared once it is set by software. It can only be cleared by a system reset. 1 All SLB bits are write protected and can not be modified 0 All SLB bits are accessible and can be modified.
UAA	User Access Allowed. 1 The registers in the module under protection can be accessed in the mode defined for the module registers without any additional restrictions. 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.

Note: The GCR.UAA bit has no effect on the allowed access modes for the registers in the Register Protection module.

29.6 Functional description

29.6.1 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)

Which addresses are protected and the protection size depend on the SoC and/or module. Therefore this section can just give examples for various protection configurations.

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 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.

29.6.2 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* 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

Memory area #4 contains the lock bits. They can be modified by writing to them. Each SLBR*n*.SLB*m* bit has a mask bit SLBR*n*.WE*m,* which protects it from being modified. This masking makes clear-modify-write operations unnecessary.

Figure 392 shows two modification examples. In the left example there is a write access to the SLBR*n* register specifying a mask value which allows modification of all SLBR*n*.SLB*m* bits. The example on the right specifies a mask which only allows modification of the bits SLBR*n*.SLB[3:1].

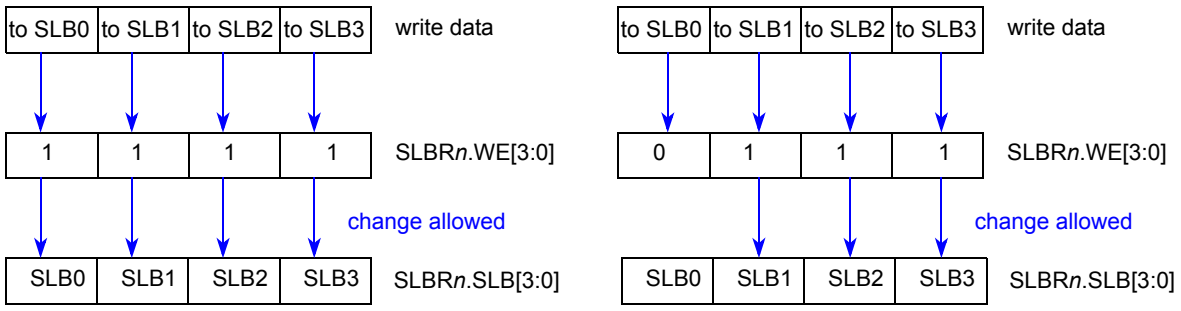


Figure 392. Change Lock Settings Directly Via Area #4

Figure 392 shows four registers that can be protected 8-bit wise. In *Figure 393* registers with 16-bit protection and in *Figure 394* registers with 32-bit protection are shown:

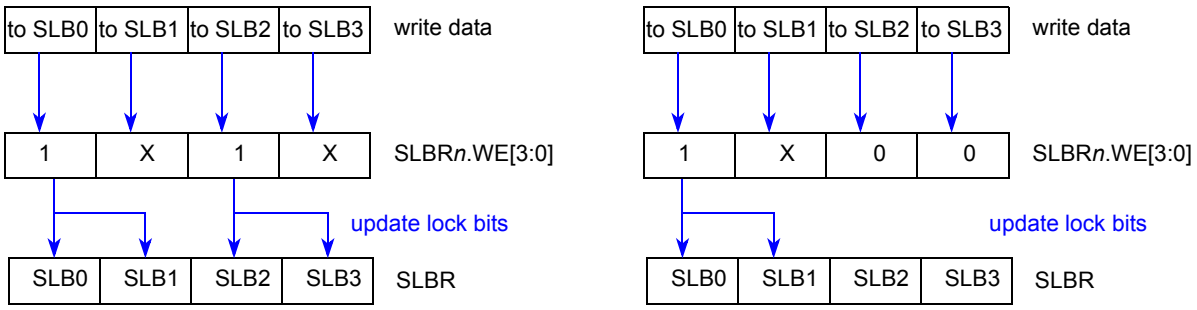


Figure 393. Change Lock Settings for 16-bit Protected Addresses

On the right side of *Figure 393* it is shown that the data written to SLBR*n*.SLB[0] is automatically written to SLBR*n*.SLB[1] also. This is done as the address reflected by SLBR*n*.SLB[0] is protected 16-bit wise. Note that in this case the write enable SLBR*n*.WE[0] must be set while SLBR*n*.WE[1] 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 393* the data written to SLBR*n*.SLB[0] is mirrored to SLBR*n*.SLB[1] and the data written to SLBR*n*.SLB[2] is mirrored to SLBR*n*.SLB[3] as for both registers the write enables are set.

In *Figure 394* a 32-bit wise protected register is shown. When SLBR*n*.WE[0] is set the data written to SLBR*n*.SLB[0] is automatically written to SLBR*n*.SLB[3:1] also. Otherwise SLBR*n*.SLB[3:0] remains unchanged.

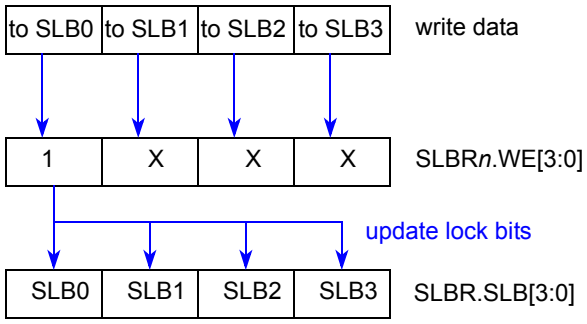


Figure 394. Change Lock Settings for 32-bit Protected Addresses

In *Figure 395* an example is shown which has a mixed protection size configuration:

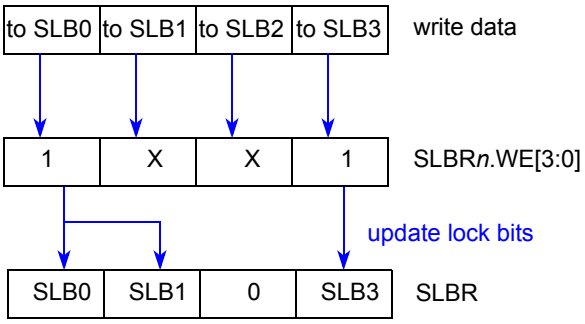


Figure 395. Change Lock Settings for Mixed Protection

The data written to SLBR*n*.SLB[0] is mirrored to SLBR*n*.SLB[1] as the corresponding register is 16-bit protected. The data written to SLBR*n*.SLB[2] is blocked as the corresponding register is unprotected. The data written to SLBR*n*.SLB[3] is written to SLBR*n*.SLB[3].

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 396* shows one example:

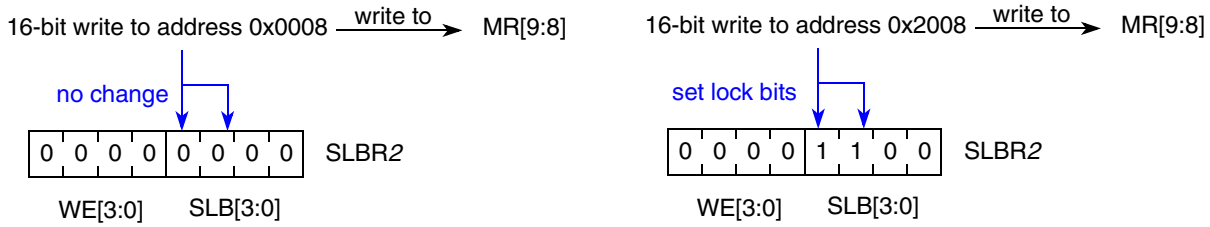


Figure 396. Enable Locking Via Mirror Module Space (Area #3)

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 393*).

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 393*).

Figure 397 shows an example where some addresses are protected and some are not:

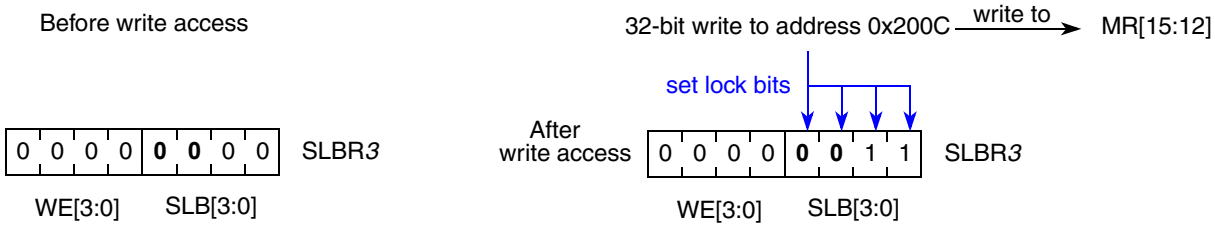


Figure 397. Enable Locking for Protected and Unprotected Addresses

In the example in *Figure 397* 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 bit GCR.HLB is cleared. Once this bit is set the locking bits can no longer be modified until there is a system reset.

29.6.3 Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to *Figure 389*.

1. If accessing area #1 or area #3, the protection module transfers any access error from the underlying Module under Protection.
2. If user mode is not allowed, user write attempts to all areas will assert a transfer error and the writes will be blocked.
3. Access attempts to the reserved area #2 cause a transfer error to be asserted.
4. Access attempts to unimplemented 32-bit registers in area #4 or area #5 cause a transfer error to be asserted.
5. Attempted writes to a register in area #1 or area #3 with soft lock bit set for any of the affected bytes causes a transfer error to be asserted and the write is blocked. The complete write operation to non-protected bytes in this word is ignored.
6. If writing to a soft lock register in area #4 with the hard lock bit being set a transfer error is asserted.
7. Any write operation in any access mode to area #3 while GCR.HLB is set result in a error.

29.7 Reset

The reset state of each individual bit is shown within the Register Description section (See *Section 29.5.2, Register description*). In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in Supervisor Mode only.

29.8 Protected registers

For SPC560Bx and SPC560Cx the Register Protection module protects the registers shown in *Table 384*.

Table 384. Protected registers

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
Code flash memory , 4 registers to protect					
Code Flash	MCR	32	C3F88000	000	bits[0:31]
Code Flash	PFCR0	32	C3F88000	01C	bits[0:31]
Code Flash	PFCR1	32	C3F88000	020	bits[0:31]
Code Flash	PFAPR	32	C3F88000	024	bits[0:31]
Data flash memory, 1 register to protect					
Data Flash	MCR	32	C3F8C000	000	bits[0:31]
SIU lite, 64 registers to protect					
SIUL	IRER	32	C3F90000	018	bits[0:31]
SIUL	IREER	32	C3F90000	028	bits[0:31]
SIUL	IFEER	32	C3F90000	02C	bits[0:31]
SIUL	IFER	32	C3F90000	030	bits[0:31]

Table 384. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
SIUL	PCR0	16	C3F90000	040	bits[0:15]
SIUL	PCR1	16	C3F90000	042	bits[0:15]
SIUL	PCR2	16	C3F90000	044	bits[0:15]
SIUL	PCR3	16	C3F90000	046	bits[0:15]
SIUL	PCR4	16	C3F90000	048	bits[0:15]
SIUL	PCR5	16	C3F90000	04A	bits[0:15]
SIUL	PCR6	16	C3F90000	04C	bits[0:15]
SIUL	PCR7	16	C3F90000	04E	bits[0:15]
SIUL	PCR8	16	C3F90000	050	bits[0:15]
SIUL	PCR9	16	C3F90000	052	bits[0:15]
SIUL	PCR10	16	C3F90000	054	bits[0:15]
SIUL	PCR11	16	C3F90000	056	bits[0:15]
SIUL	PCR12	16	C3F90000	058	bits[0:15]
SIUL	PCR13	16	C3F90000	05A	bits[0:15]
SIUL	PCR14	16	C3F90000	05C	bits[0:15]
SIUL	PCR15	16	C3F90000	05E	bits[0:15]
SIUL	PCR16	16	C3F90000	060	bits[0:15]
SIUL	PCR17	16	C3F90000	062	bits[0:15]
SIUL	PCR18	16	C3F90000	064	bits[0:15]
SIUL	PCR19	16	C3F90000	066	bits[0:15]
SIUL	PCR34	16	C3F90000	084	bits[0:15]
SIUL	PCR35	16	C3F90000	086	bits[0:15]
SIUL	PCR36	16	C3F90000	088	bits[0:15]
SIUL	PCR37	16	C3F90000	08A	bits[0:15]
SIUL	PCR38	16	C3F90000	08C	bits[0:15]
SIUL	PCR39	16	C3F90000	08E	bits[0:15]
SIUL	PCR40	16	C3F90000	090	bits[0:15]
SIUL	PCR41	16	C3F90000	092	bits[0:15]
SIUL	PCR42	16	C3F90000	094	bits[0:15]
SIUL	PCR43	16	C3F90000	096	bits[0:15]
SIUL	PCR44	16	C3F90000	098	bits[0:15]
SIUL	PCR45	16	C3F90000	09A	bits[0:15]

Table 384. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
SIUL	PCR46	16	C3F90000	09C	bits[0:15]
SIUL	PCR47	16	C3F90000	09E	bits[0:15]
SIUL	PSMI0	8	C3F90000	500	bits[0:7]
SIUL	PSMI4	8	C3F90000	504	bits[0:7]
SIUL	PSMI8	8	C3F90000	508	bits[0:7]
SIUL	PSMI12	8	C3F90000	50C	bits[0:7]
SIUL	PSMI16	8	C3F90000	510	bits[0:7]
SIUL	IFMC0	32	C3F90000	1000	bits[0:31]
SIUL	IFMC1	32	C3F90000	1004	bits[0:31]
SIUL	IFMC2	32	C3F90000	1008	bits[0:31]
SIUL	IFMC3	32	C3F90000	100C	bits[0:31]
SIUL	IFMC4	32	C3F90000	1010	bits[0:31]
SIUL	IFMC5	32	C3F90000	1014	bits[0:31]
SIUL	IFMC6	32	C3F90000	1018	bits[0:31]
SIUL	IFMC7	32	C3F90000	101C	bits[0:31]
SIUL	IFMC8	32	C3F90000	1020	bits[0:31]
SIUL	IFMC9	32	C3F90000	1024	bits[0:31]
SIUL	IFMC10	32	C3F90000	1028	bits[0:31]
SIUL	IFMC11	32	C3F90000	102C	bits[0:31]
SIUL	IFMC12	32	C3F90000	1030	bits[0:31]
SIUL	IFMC13	32	C3F90000	1034	bits[0:31]
SIUL	IFMC14	32	C3F90000	1038	bits[0:31]
SIUL	IFMC15	32	C3F90000	103C	bits[0:31]
SIUL	IFCPR	32	C3F90000	1080	bits[0:31]
Mode Entry Module, 41 registers to protect					
MC ME	ME_ME	32	C3FDC000	008	bits[0:31]
MC ME	ME_IM	32	C3FDC000	010	bits[0:31]
MC ME	ME_TEST_MC	32	C3FDC000	024	bits[0:31]
MC ME	ME_SAFE_MC	32	C3FDC000	028	bits[0:31]
MC ME	ME_DRUN_MC	32	C3FDC000	02C	bits[0:31]
MC ME	ME_RUN0_MC	32	C3FDC000	030	bits[0:31]
MC ME	ME_RUN1_MC	32	C3FDC000	034	bits[0:31]

Table 384. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
MC ME	ME_RUN2_MC	32	C3FDC000	038	bits[0:31]
MC ME	ME_RUN3_MC	32	C3FDC000	03C	bits[0:31]
MC ME	ME_HALT_MC	32	C3FDC000	040	bits[0:31]
MC ME	ME_STOP_MC	32	C3FDC000	048	bits[0:31]
MC ME	ME_STANDBY_MC	32	C3FDC000	054	bits[0:31]
MC ME	ME_RUN_PC0	32	C3FDC000	080	bits[0:31]
MC ME	ME_RUN_PC1	32	C3FDC000	084	bits[0:31]
MC ME	ME_RUN_PC2	32	C3FDC000	088	bits[0:31]
MC ME	ME_RUN_PC3	32	C3FDC000	08C	bits[0:31]
MC ME	ME_RUN_PC4	32	C3FDC000	090	bits[0:31]
MC ME	ME_RUN_PC5	32	C3FDC000	094	bits[0:31]
MC ME	ME_RUN_PC6	32	C3FDC000	098	bits[0:31]
MC ME	ME_RUN_PC7	32	C3FDC000	09C	bits[0:31]
MC ME	ME_LP_PC0	32	C3FDC000	0A0	bits[0:31]
MC ME	ME_LP_PC1	32	C3FDC000	0A4	bits[0:31]
MC ME	ME_LP_PC2	32	C3FDC000	0A8	bits[0:31]
MC ME	ME_LP_PC3	32	C3FDC000	0AC	bits[0:31]
MC ME	ME_LP_PC4	32	C3FDC000	0B0	bits[0:31]
MC ME	ME_LP_PC5	32	C3FDC000	0B4	bits[0:31]
MC ME	ME_LP_PC6	32	C3FDC000	0B8	bits[0:31]
MC ME	ME_LP_PC7	32	C3FDC000	0BC	bits[0:31]
MC ME	ME_PCTL[4..7]	32	C3FDC000	0C4	bits[0:31]
MC ME	ME_PCTL[16..19]	32	C3FDC000	0D0	bits[0:31]
MC ME	ME_PCTL[20..23]	32	C3FDC000	0D4	bits[0:31]
MC ME	ME_PCTL[32..35]	32	C3FDC000	0E0	bits[0:31]
MC ME	ME_PCTL[44..47]	32	C3FDC000	0EC	bits[0:31]
MC ME	ME_PCTL[48..51]	32	C3FDC000	0F0	bits[0:31]
MC ME	ME_PCTL[56..59]	32	C3FDC000	0F8	bits[0:31]
MC ME	ME_PCTL[60..63]	32	C3FDC000	0FC	bits[0:31]
MC ME	ME_PCTL[68..71]	32	C3FDC000	104	bits[0:31]
MC ME	ME_PCTL[72..75]	32	C3FDC000	108	bits[0:31]
MC ME	ME_PCTL[88..91]	32	C3FDC000	118	bits[0:31]

Table 384. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
MC ME	ME_PCTL[92..95]	32	C3FDC000	11C	bits[0:31]
MC ME	ME_PCTL[104..107]	32	C3FDC000	128	bits[0:31]
Clock Generation Module, 3 registers to protect					
MC CGM	CGM_OC_EN	8	C3FE0000	373	bits[0:7]
MC CGM	CGM_OCDS_SC	8	C3FE0000	374	bits[0:7]
MC CGM	CGM_SC_DC[0..3]	32	C3FE0000	37C	bits[0:31]
CMU, 1 register to protect					
CMU	CMU_CSR	8	C3FE0100	000	bits[24:31]
Reset Generation Module, 7 registers to protect					
MC RGM	RGM_FERD	16	C3FE4000	004	bits[0:15]
MC RGM	RGM_DERD	16	C3FE4000	006	bits[0:15]
MC RGM	RGM_FEAR	16	C3FE4000	010	bits[0:15]
MC RGM	RGM_DEAR	16	C3FE4000	012	bits[0:15]
MC RGM	RGM_FESS	16	C3FE4000	018	bits[0:15]
MC RGM	RGM_STDBY	16	C3FE4000	01A	bits[0:15]
MC RGM	RGM_FBRE	16	C3FE4000	01C	bits[0:15]
Power Control Unit, 1 registers to protect					
MC PCU	PCONF2	32	C3FE8000	008	bits[0:31]

30 Software Watchdog Timer (SWT)

30.1 Overview

The 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 generates a reset.

30.2 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 slow internal RC oscillator 128 kHz (SIRC), 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 phase3 and optionally enabled when platform reset is released depending on value of flash user option bit 31 (WATCHDOG_EN).

30.3 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. On exit from STOP mode, the SWT will continue from the state it was before entering this mode.

The software watchdog is not available during standby. On exit from standby, the SWT behaves in a usual “out of reset” situation.

30.4 External signal description

The SWT module does not have any external interface signals.

30.5 Memory map and register description

The SWT programming model has six 32-bit registers. 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. A bus error is generated on invalid accesses. If the SWT_CR[RIA] bit is set, then the SWT system reset is also 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.

30.5.1 Memory map

The SWT memory map is shown in *Table 385*. The reset values of SWT_CR, SWT_TO and SWT_WN are device specific. These values are determined by SWT inputs.

Table 385. SWT memory map

Base address: 0xFFF3_8000		
Address offset	**Register**	**Location**
0x0000	SWT Control Register (SWT_CR)	*on page 30-738*
0x0004	SWT Interrupt Register (SWT_IR)	*on page 30-740*
0x0008	SWT Time-Out Register (SWT_TO)	*on page 30-741*
0x000C	SWT Window Register (SWT_WN)	*on page 30-741*
0x0010	SWT Service Register (SWT_SR)	*on page 30-742*
0x0014	SWT Counter Output Register (SWT_CO)	*on page 30-742*

30.5.2 Register description

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 398. SWT Control Register (SWT_CR)

Offset 0x0000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP 0	MAP 1	MAP 2	MAP 3	MAP 4	MAP 5	MAP 6	MAP 7	0	0	0	0	0	0	0	0
W																
Reset(1)	0	1	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	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset(1)	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1

1. The reset value for the SWT_CR is device specific.

Default value for SWT_CR_RST is 0x4000_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'.

Table 386. SWT_CR field descriptions

Field	Description
MAPn	Master Access Protection for Master n. The platform bus master assignments are device specific. 0 = Access for the master is not enabled 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 value 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

Table 386. SWT_CR field descriptions (continued)

Field	Description
CSL	Clock Selection. Selects the SIRC 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 SPC560Bx and SPC560Cx device. 0 = System clock (Not applicable in SPC560Bx and SPC560Cx) 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 399. SWT Interrupt Register (SWT_IR)

Offset 0x0004 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	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

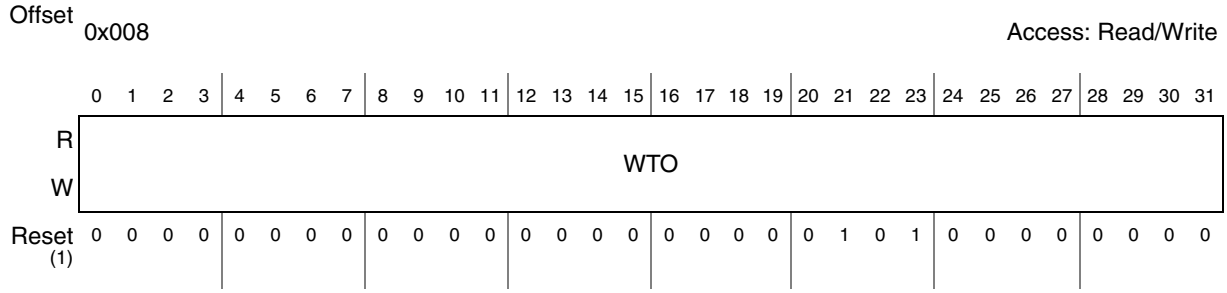
Table 387. SWT_IR field descriptions

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 400. SWT Time-Out Register (SWT_TO)



1. The reset value of the SWT_TO register is device specific.

Default counter value (SWT_TO_RST) is 1280 (0x00000500 hexadecimal) which correspond to around 10 ms with a 128 kHz clock.

Table 388. SWT_TO Register field descriptions

Field	Description
WTO	Watchdog time-out period in clock cycles. An internal 32-bit down counter is loaded with this value or 0x100 which ever 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 401. SWT Window Register (SWT_WN)

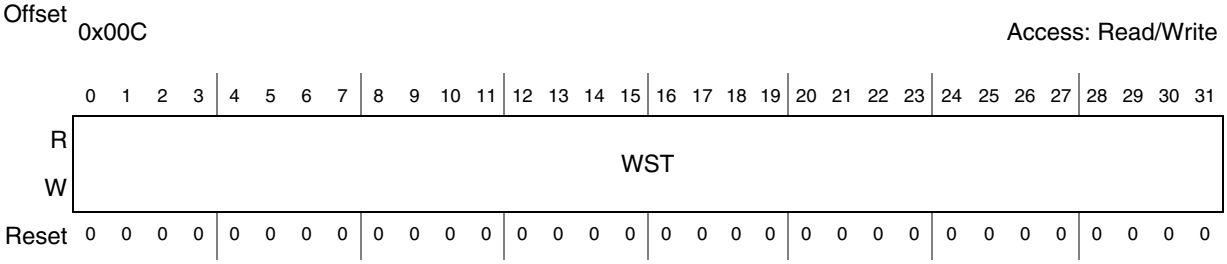


Table 389. SWT_WN Register field descriptions

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 402. SWT Service Register (SWT_SR)

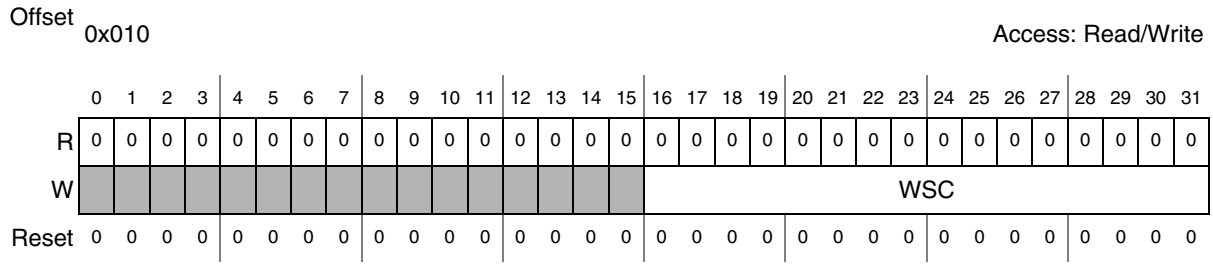


Table 390. SWT_SR field descriptions

Field	Description
WSC	Watchdog Service Code.This field is used to service the watchdog and to clear the soft lock bit (SWT_CR.SLK). To service the watchdog, the value 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR.SLKSWT_CR.), 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 403. SWT Counter Output Register (SWT_CO)

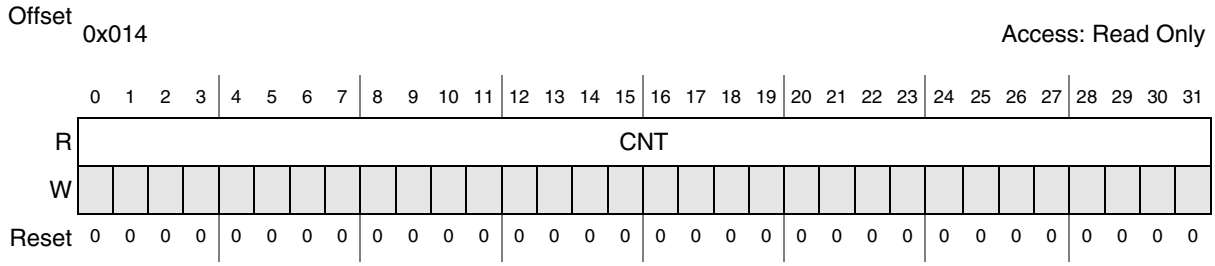


Table 391. SWT_CO field descriptions

Field	Description
CNT	Watchdog Count. When the watchdog is disabled (SWT_CR.WENSWT_CR.=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 up to six system plus eight 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.

30.6 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 a control register (SWT_CR), an interrupt register (SWT_IR), time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR) and a counter output register (SWT_CO).

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 specific1 (enabled). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG_EN) is ‘0’. 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 sequence is written. 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 as described previously.

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 and SWT_WN 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 the watchdog servicing sequence. The service sequence is a write of 0xA602 followed by a write of 0xB480 to the SWT_SR.WSC field. Writing the service sequence loads the internal down counter with the time-out period. There is no timing requirement between the two writes. The service sequence logic ignores unlock sequence writes and recognizes the 0xA602, 0xB480 sequence regardless of previous writes. 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.

Note: Watchdog is disabled at the start of BAM execution. In the case of an unexpected issue during BAM execution, the CPU may be stalled and an external reset needs to be generated to recover.

31 Error Correction Status Module (ECSM)

31.1 Introduction

The 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, and information on memory errors reported by error-correcting codes.

31.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of miscellaneous control functions for the device.

31.3 Features

The ECSM includes these features:

- Program-visible information on the device configuration and revision
- Registers for capturing information on memory errors due to error-correction codes
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes to check ECC protection
- Configuration for additional SRAM WS for system frequency above 64 + 4% MHz

31.4 Memory map and register description

This section details the programming model for the Error Correction Status Module. This is a 128-byte space mapped to the region serviced by an IPS bus controller.

31.4.1 Memory map

The Error Correction Status Module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

Table 392 shows the ECSM's memory map.

Table 392. ECSM memory map

Base address: 0xFFF4_0000		
Address offset	**Register**	**Location**
0x00	Processor Core Type Register (PCT)	*on page 31-747*
0x02	SoC-Defined Platform Revision Register (REV)	*on page 31-747*
0x04	Reserved	
0x08	IPS On-Platform Module Configuration Register (IOPMC)	*on page 31-747*
0x0C–0x12	Reserved	

Table 392. ECSM memory map (continued)

Base address: 0xFFF4_0000		
Address offset	**Register**	**Location**
0x13	Miscellaneous Wakeup Control Register (MWCR)	*on page 31-748*
0x14–0x1E	Reserved	
0x1F	Miscellaneous Interrupt Register (MIR)	*on page 31-750*
0x20–0x23	Reserved	
0x24	Miscellaneous User-Defined Control Register (MUDCR)	*on page 31-751*
0x28–0x42	Reserved	
0x43	ECC Configuration Register (ECR)	*on page 31-752*
0x44–0x46	Reserved	
0x47	ECC Status Register (ESR)	*on page 31-754*
0x48–0x49	Reserved	
0x4A	ECC Error Generation Register (EEGR)	*on page 31-756*
0x4C–0x4F	Reserved	
0x50	Platform Flash ECC Address Register (PFEAR)	*on page 31-758*
0x54–0x55	Reserved	
0x56	Platform Flash ECC Master Number Register (PFEMR)	*on page 31-760*
0x57	Platform Flash ECC Attributes Register (PFEAT)	*on page 31-760*
0x58–0x5B	Reserved	
0x5C	Platform Flash ECC Data Register (PFEDR)	*on page 31-761*
0x60	Platform RAM ECC Address Register (PREAR)	*on page 31-762*
0x64	Reserved	
0x65	Platform RAM ECC Syndrome Register (PRESR)	*on page 31-762*
0x66	Platform RAM ECC Master Number Register (PREMR)	*on page 31-764*
0x67	Platform RAM ECC Attributes Register (PREAT)	*on page 31-765*
0x68–0x6B	Reserved	
0x6C	Platform RAM ECC Data Register (PREDR)	*on page 31-766*

31.4.2 Register 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 Register (PCT)

The PCT is a 16-bit read-only register specifying 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 404. Processor Core Type Register (PCT)

Offset: 0x00 Access: Read

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

Table 393. PCT field descriptions

Field	Description
PCT	**Processor Core Type**

SoC-Defined Platform Revision Register (REV)

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 405. SoC-Defined Platform Revision Register (REV)

Offset: 0x02 Access: Read

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

Table 394. REV field descriptions

Field	Description
REV	**Revision** The REV field is specified by an input signal to define a software-visible revision number.

IPS On-Platform Module Configuration Register (IOPMC)

The IOPMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary IPI 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 406. IPS On-Platform Module Configuration Register (IOPMC)

Offset: 0x08 Access: Read

	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	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MC[15:0]															
W																
Reset:	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1

Table 395. IOPMC field descriptions

Field	Description
MC	**IPS Module Configuration** MC[n] = 0 if an IPS module connection to decoded slot “n” is absent MC[n] = 1 if an IPS module connection to decoded slot “n” is present

Miscellaneous Wakeup Control Register (MWCR)

Implementation of low-power sleep modes and exit from these modes via an interrupt require communication between the ECSM, the interrupt controller and off-platform external logic typically associated with phase-locked loop clock generation circuitry. The Miscellaneous Wakeup Control Register (MWCR) provides an 8-bit register controlling entry into these types of low-power modes as well as definition of the interrupt level needed to exit the mode.

The following sequence of operations is generally needed to enable this functionality. Note that the exact details are likely to be system-specific.

1. The processor core loads the appropriate data value into the MWCR, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor ceases execution. The exact mechanism varies by processor core. In some cases, a processor-is-stopped status is signaled to the ECSM and off-platform external logic. This assertion, if properly enabled by MWCR[ENBWCR], causes the ECSM output signal “enter_low_power_mode” to be set. This, in turn, causes the selected off-platform external, low-power mode, as specified by MWCR[LPMD], to be entered, and the appropriate clock signals disabled. In most implementations, there are multiple low-power modes, where the exact clocks to be disabled vary across the different modes.
3. After entering the low-power mode, the interrupt controller enables a special combinational logic path which evaluates all unmasked interrupt requests. The device

remains in this mode until an event which generates an unmasked interrupt request with a priority level greater than the value programmed in the MWCR[PRILVL] occurs.

4. Once the appropriately-high interrupt request level arrives, the interrupt controller signals its presence, and the ECSM responds by asserting an “exit_low_power_mode” signal.
5. The off-platform external logic senses the assertion of the “exit” signal, and re-enables the appropriate clock signals.
6. With the processor core clocks enabled, the core handles the pending interrupt request.

Figure 407. Miscellaneous Wakeup Control (MWCR) Register

Offset: 0x13 Access: Read/write

	0	1	2	3	4	5	6	7
R	ENBWCR	0	0	0	PRILVL			
W								
Reset:	0	0	0	0	0	0	0	0

Table 396. MWCR field descriptions

Field	Description
ENBWCR	**Enable WCR** 0 MWCR is disabled. 1 MWCR is enabled.
PRILVL	**Interrupt Priority Level** The interrupt priority level is a core-specific definition. It specifies the interrupt priority level needed to exit the low-power mode. Specifically, an unmasked interrupt request of a priority level greater than the PRILVL value is required to exit the mode. Certain interrupt controller implementations include logic associated with this priority level that restricts the data value contained in this field to a [0, maximum - 1] range. See the specific interrupt controller module for details.

Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the MIR. This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the MIR must be explicitly cleared. See *Figure 408* and *Table 397*.

Figure 408. Miscellaneous Interrupt (MIR) Register

Offset: 0x1F Access: Special

	0	1	2	3	4	5	6	7
R	FB0AI	FB0SI	FB1AI	FB1SI	0	0	0	0
W	1	1	1	1				
Reset:	0	0	0	0	0	0	0	0

Table 397. MIR field descriptions

Field	Description
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.
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.
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.
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 typically is used as configuration control for miscellaneous SoC-level modules. The contents of this register is simply output from the ECSM to other modules where the user-defined control functions are implemented.

Figure 409. Miscellaneous User-Defined Control (MUDCR) Register

Offset: 0x24 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MUDCR[31]	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

Table 398. MUDCR field descriptions

Field	Description
MUDCR[31]	**XBAR force_round_robin bit** This bit is used to drive the force_round_robin bit of the XBAR. This will force the slaves into round robin mode of arbitration rather than fixed mode (unless a master is using priority elevation, which forces the design back into fixed mode regardless of this bit). By setting the hardware definition to ENABLE_ROUND_ROBIN_RESET, this bit will reset to 1. 1 XBAR is in round robin mode 0 XBAR is in fixed priority mode

ECC registers

For designs including error-correcting code (ECC) implementations to improve the quality and reliability of memories, there are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Platform Flash ECC Address Register (PFEAR)
- Platform Flash ECC Master Number Register (PFEMR)
- Platform Flash ECC Attributes Register (PFEAT)
- Platform Flash ECC Data Register (PFEDR)
- Platform RAM ECC Address Register (PREAR)
- Platform RAM ECC Syndrome Register (PRESR)
- Platform RAM ECC Master Number Register (PREMR)
- Platform RAM ECC Attributes Register (PREAT)
- Platform RAM ECC Data Register (PREDR)

The details on the ECC registers are provided in the subsequent sections.

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.) which may be useful for subsequent failure analysis.

Figure 410. ECC Configuration (ECR) Register

Offset: 0x43 Access: 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 399. ECR field descriptions

Field	Description
ER1BR	**Enable SRAM 1-bit Reporting** The occurrence of a single-bit SRAM correction generates a ECSM ECC interrupt request as signalled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers. 0 Reporting of single-bit SRAM corrections is disabled. 1 Reporting of single-bit SRAM corrections is enabled.
EF1BR	**Enable Flash 1-bit Reporting** The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signalled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers. 0 Reporting of single-bit flash corrections is disabled. 1 Reporting of single-bit flash corrections is enabled.
ERNCR	**Enable SRAM Non-Correctable Reporting** The occurrence of a non-correctable multi-bit SRAM error generates a ECSM ECC interrupt request as signalled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers. 0 Reporting of non-correctable SRAM errors is disabled. 1 Reporting of non-correctable SRAM errors is enabled.
EFNCR	**Enable Flash Non-Correctable Reporting** The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signalled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers. 0 Reporting of non-correctable flash errors is disabled. 1 Reporting of non-correctable flash errors is 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 411. ECC Status Register (ESR)

Offset: 0x47 Access: 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 400. ESR field descriptions

Field	Description
R1BC	**SRAM 1-bit Correction** This bit can only be set if ECR[EPR1BR] is asserted. The occurrence of a properly-enabled single-bit SRAM correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit SRAM correction has been detected. 1 A reportable single-bit SRAM correction has been detected.
F1BC	**Flash Memory 1-bit Correction** This bit can only be set if ECR[EPF1BR] is asserted. The occurrence of a properly-enabled single-bit flash memory correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit flash memory correction has been detected. 1 A reportable single-bit flash memory correction has been detected.
RNCE	**SRAM Non-Correctable Error** The occurrence of a properly-enabled non-correctable SRAM error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable SRAM error has been detected. 1 A reportable non-correctable SRAM error has been detected.
FNCE	**Flash Memory Non-Correctable Error** The occurrence of a properly-enabled non-correctable flash memory error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable flash memory error has been detected. 1 A reportable non-correctable flash memory error has been 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 SRAM. 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 SRAM, 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 (SRAM 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.

Figure 412. ECC Error Generation Register (EEGR)

Offset: 0x4A Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FRC1BI	FR11BI	0	0	FRCNCI	FR1NCI	0	ERRBIT						
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 401. EEGR field descriptions

Field	Description
FRC1BI	**Force SRAM Continuous 1-bit Data Inversions** The assertion of this bit forces the SRAM 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 SRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the SRAM. 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 SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted. 0 No SRAM continuous 1-bit data inversions are generated. 1 1-bit data inversions in the SRAM are continuously generated.
FR11BI	**Force SRAM One 1-bit Data Inversion** The assertion of this bit forces the SRAM 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 SRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the SRAM. 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 SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted. 0 No SRAM single 1-bit data inversion is generated. 1 One 1-bit data inversion in the SRAM is generated.
FRCNCI	**Force SRAM Continuous Non-correctable Data Inversions** The assertion of this bit forces the SRAM 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 SRAM 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 SRAM. 0 No SRAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the SRAM are continuously generated.

Table 401. EEGR field descriptions (continued)

Field	Description
FR1NCI	**Force SRAM One Non-correctable Data Inversions** The assertion of this bit forces the SRAM 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 SRAM 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 SRAM. 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. 0 No SRAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the SRAM is generated.
ERRBIT	**Error Bit Position** The vector defines the bit position which 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 SRAM 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 SRAM width. For example, consider a 32-bit SRAM 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 (32b data + 7b for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then SRAM[0] of the odd bank is inverted if ERRBIT = 1, then SRAM[1] of the odd bank is inverted ... if ERRBIT = 31, then SRAM[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.

Platform Flash ECC Address Register (PFEAR)

The PFEAR 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 PFEAR, PFEMR, PFEAT and PFEDR 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 413. Platform Flash ECC Address Register (PFEAR)

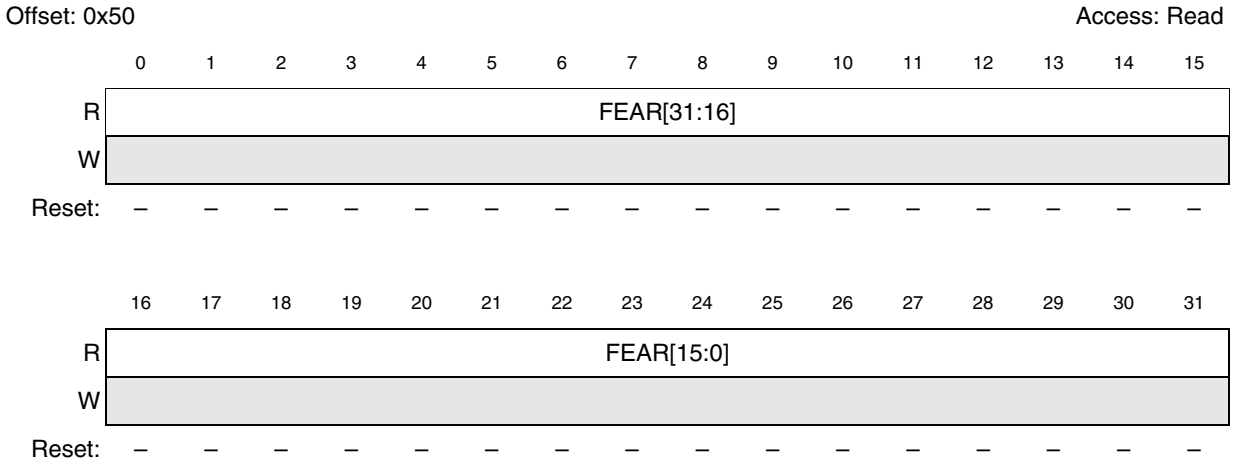


Table 402. PFEAR field descriptions

Field	Description
FEAR	**Flash ECC Address Register** This 32-bit register contains the faulting access address of the last, properly-enabled flash ECC event.

Platform Flash ECC Master Number Register (PFEMR)

The PFEMR 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 PFEAR, PFEMR, PFEAT and PFEDR 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 414. Platform Flash ECC Master Number Register (PFEMR)

Offset: 0x56 Access: Read

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR			
W								
Reset:	0	0	0	0	–	–	–	–

Table 403. PFEMR field descriptions

Field	Description
FEMR	**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.

Platform Flash ECC Attributes Register (PFEAT)

The PFEAT 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 PFEAR, PFEMR, PFEAT and PFEDR 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 415. Platform Flash ECC Attributes Register (PFEAT)

Offset: 0x57 Access: Read

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset:	–	–	–	–	–	–	–	–

Table 404. PFEAT field descriptions

Field	Description
WRITE	**AMBA-AHB HWRITE** 0 AMBA-AHB read access 1 AMBA-AHB write access
SIZE	**AMBA-AHB HSIZE[2:0]** 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1xx Reserved
PROTECTION	**AMBA-AHB HPROT[3:0]** Protection[3]: Cacheable — 0 = Non-cacheable, 1 = Cacheable Protection[2]: Bufferable — 0 = Non-bufferable, 1 = Bufferable Protection[1]: Mode — 0 = User mode, 1 = Supervisor mode Protection[0]: Type — 0 = I-Fetch, 1 = Data

Platform Flash ECC Data Register (PFEDR)

The PFEDR 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 PFEAR, PFEMR, PFEAT and PFEDR 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 416. Platform Flash ECC Data Register (PFEDR)

Offset: 0x5C Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEDR[31:16]															
W																
Reset:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 405. PFEDR field descriptions

Field	Description
FEDR	**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.

Platform RAM ECC Address Register (PREAR)

The PREAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR 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 417. Platform RAM ECC Address Register (PREAR)

Offset: 0x60 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REAR[31:16]															
W																
Reset:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 406. PREAR field descriptions

Field	Description
REAR	**SRAM ECC Address Register** This 32-bit register contains the faulting access address of the last, properly-enabled SRAM ECC event.

Platform RAM ECC Syndrome Register (PRESR)

The PRESR is an 8-bit register for capturing the error syndrome of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR 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 418. Platform RAM ECC Syndrome Register (PRESR)

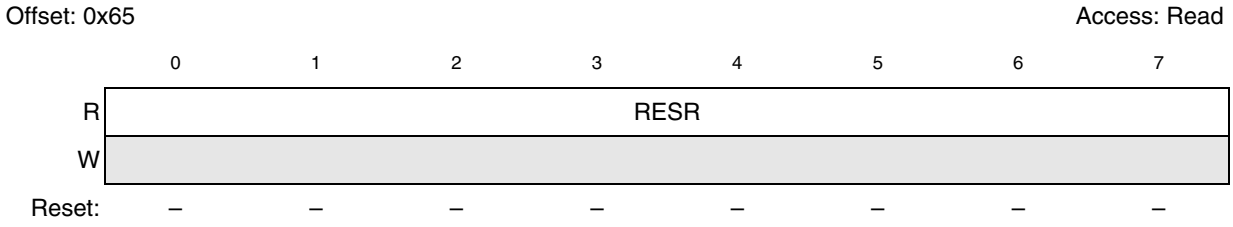


Table 407. PRESR field descriptions

Field	Description
RESR	**SRAM 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 408* associates the upper 7 bits of the syndrome with the data bit in error.

Table 408 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 0x01 implies no error condition but this value is not readable when the PRESR is read for the no error case.

Table 408. RAM syndrome mapping for single-bit correctable errors

PRESR[RESR]	Data bit in error
0x00	ECC ODD[0]
0x01	No error
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x06	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x0a	DATA ODD BANK[30]
0x0c	DATA ODD BANK[29]
0x0e	DATA ODD BANK[28]
0x10	ECC ODD[4]
0x12	DATA ODD BANK[27]
0x14	DATA ODD BANK[26]
0x16	DATA ODD BANK[25]

Table 408. RAM syndrome mapping for single-bit correctable errors (continued)

PRESR[RESR]	Data bit in error
0x18	DATA ODD BANK[24]
0x1a	DATA ODD BANK[23]
0x1c	DATA ODD BANK[22]
0x50	DATA ODD BANK[21]
0x20	ECC ODD[5]
0x22	DATA ODD BANK[20]
0x24	DATA ODD BANK[19]
0x26	DATA ODD BANK[18]
0x28	DATA ODD BANK[17]
0x2a	DATA ODD BANK[16
0x2c	DATA ODD BANK[15]
0x58	DATA ODD BANK[14]
0x30	DATA ODD BANK[13]
0x32	DATA ODD BANK[12]
0x34	DATA ODD BANK[11]
0x64	DATA ODD BANK[10]
0x38	DATA ODD BANK[9]
0x62	DATA ODD BANK[8]
0x70	DATA ODD BANK[7]
0x60	DATA ODD BANK[6]
0x40	ECC ODD[6]
0x42	DATA ODD BANK[5]
0x44	DATA ODD BANK[4]
0x46	DATA ODD BANK[3]
0x48	DATA ODD BANK[2]
0x4a	DATA ODD BANK[1]
0x4c	DATA ODD BANK[0]
0x03,0x05.....0x4d	Multiple bit error
> 0x4d	Multiple bit error

Platform RAM ECC Master Number Register (PREMR)

The PREMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and

PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

See the XBAR chapter of this reference manual for a listing of XBAR bus master numbers.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 419. Platform RAM ECC Master Number Register (PREMR)

Offset: 0x66 Access: Read

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR			
W								
Reset:	0	0	0	0	–	–	–	–

Table 409. PREMR field descriptions

Field	Description
REMR	**SRAM ECC Master Number Register** This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly-enabled SRAM ECC event. See the XBAR chapter of this reference manual for a listing of XBAR bus master numbers.

Platform RAM ECC Attributes Register (PREAT)

The PREAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

Figure 420. Platform RAM ECC Attributes Register (PREAT)

Offset: 0x67 Access: Read

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset:	–	–	–	–	–	–	–	–

Table 410. PREAT field descriptions

Field	Description
WRITE	**XBAR HWRITE** 0 XBAR read access 1 XBAR write access
SIZE	**XBAR HSIZE[2:0]** 000 8-bit XBAR access 001 16-bit XBAR access 010 32-bit XBAR access 1xx Reserved
PROTECTION	**XBAR HPROT[3:0]** Protection[3]: Cacheable 0 = Non-cacheable, 1 = Cacheable Protection[2]: Bufferable 0 = Non-bufferable,1 = Bufferable Protection[1]: Mode 0 = User mode, 1 = Supervisor mode Protection[0]: Type 0 = I-Fetch, 1 = Data

Platform RAM ECC Data Register (PREDR)

The PREDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR 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.

Figure 421. Platform RAM ECC Data Register (PREDR)

Offset: 0x6C Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REDR[31:16]															
W																
Reset:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 411. PREDR field descriptions

Field	Description
REDR	**SRAM ECC Data Register** This 32-bit register contains the data associated with the faulting access of the last, properly-enabled SRAM ECC event. The register contains the data value taken directly from the data bus.

31.4.3 Register protection

Logic exists which restricts accesses to INTC, ECSM, MPU, STM and SWT to supervisor mode only. Accesses in User mode are not possible.

32 IEEE 1149.1 Test Access Port Controller (JTAGC)

32.1 Introduction

The JTAG port of the device consists of three inputs and one output. These pins include test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

Support of IEEE 1149.7 (cJTAGC) is planned but not actually supported on this device. for more information, please contact your sales representative.

32.2 Block diagram

Figure 422 is a block diagram of the JTAG Controller (JTAGC) block.

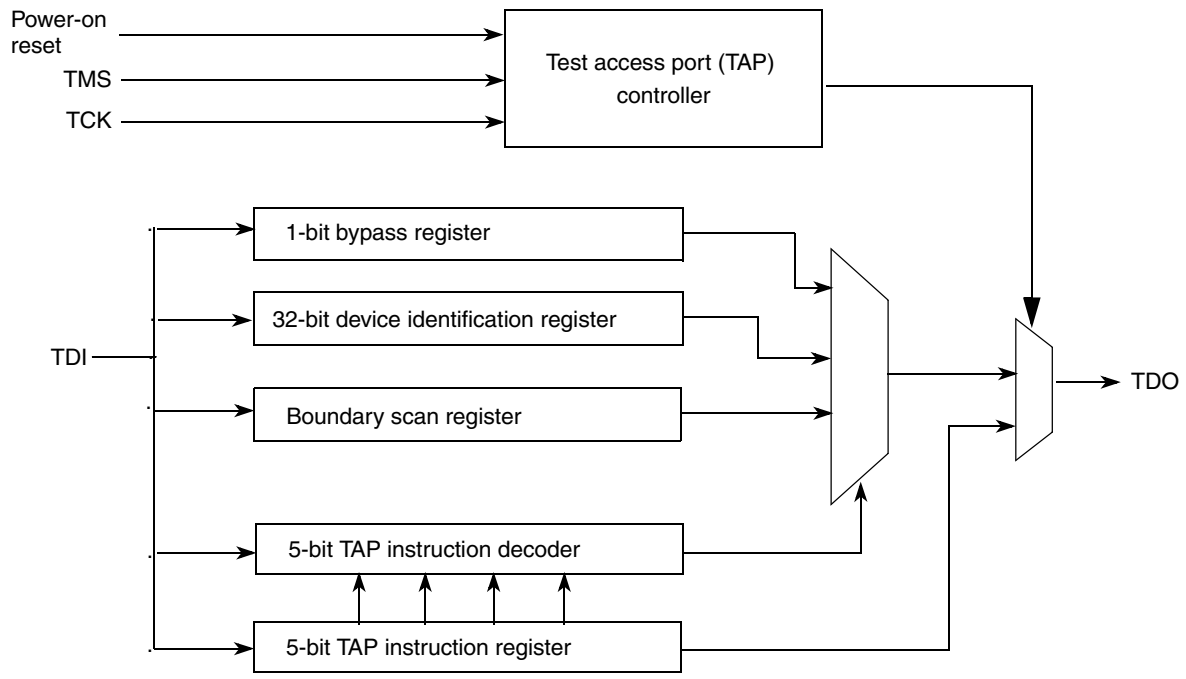


Figure 422. JTAG Controller Block Diagram

32.3 Overview

The JTAGC provides the means to test chip 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.

32.4 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
- 4 pins (TDI, TMS, TCK, and TDO)—Refer to *Section 32.6 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
- 2 test data registers:
  - Bypass register
  - Device identification register
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry

32.5 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.

32.5.1 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.

32.5.2 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, 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 external to JTAGC but can be accessed by JTAGC TAP through EXTEST,SAMPLE,SAMPLE/PRELOAD instructions. The functionality of each TEST mode is explained in more detail in *Section 32.8.4 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 is used to provide a minimum-length serial path to shift data between TDI and TDO.

TAP sharing mode

There are three selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC) and PLATFORM. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_ONCE, ACCESS_AUX_TAP_TCU. Instruction opcodes for each instruction are shown in *Table 414*.

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 the Nexus port controller chapter of the reference manual.

32.6 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 412*:

Table 412. JTAG signal properties

Name	I/O	Function	Reset State
TCK	I	Test clock	Pull Up
TDI	I	Test data in	Pull Up
TDO	O	Test data out	High Z
TMS	I	Test mode select	Pull Up

The JTAGC pins are shared with GPIO. TDO at reset is a input pad and output direction control from JTAGC. Once TAP enters shift-ir or shift-dr then output direction control from JTAGC which allows the value to see on pad. It is up to the user to configure them as GPIOs accordingly, in this case SPC560Bx and SPC560Cx get incompliance with IEEE 1149.1-2001.

32.7 Memory map and register 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.

32.7.1 Instruction Register

The JTAGC uses a 5-bit instruction register as shown in *Table 423*. 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 423. 5-bit Instruction Register

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset	0	0	0	0	1

32.7.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, 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.

32.7.3 Device Identification Register

The device identification register, shown in *Table 424*, 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 424. Device Identification Register

IR[4:0]: 0_0001 (IDCODE) Access: R/O

	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 1 0 0	0 0 0 1	0 0 0 0	0 1 0 0	0 0 0 1

Table 413. 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 SPC560Bx and SPC560Cx this value is 0x2B.
10–19 PIN	Part identification number. Contains the part number of the device. For the SPC560Bx and SPC560Cx, this value is 0x241.

Table 413. Device Identification Register Field Descriptions (continued)

Field	Description
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.

32.7.4 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select 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 32.8.5 Boundary Scan*. The size of the boundary scan register is 464 bits.

32.8 Functional Description

32.8.1 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.

32.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 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 425*. This applies for the instruction register, test data registers, and the bypass register.

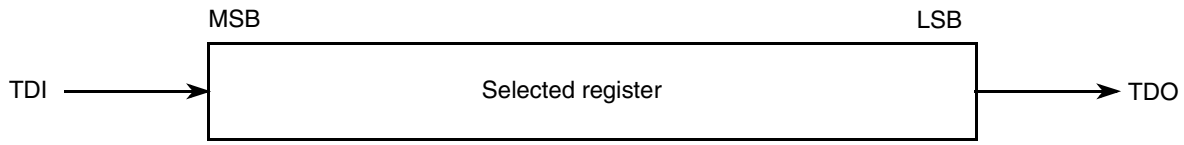
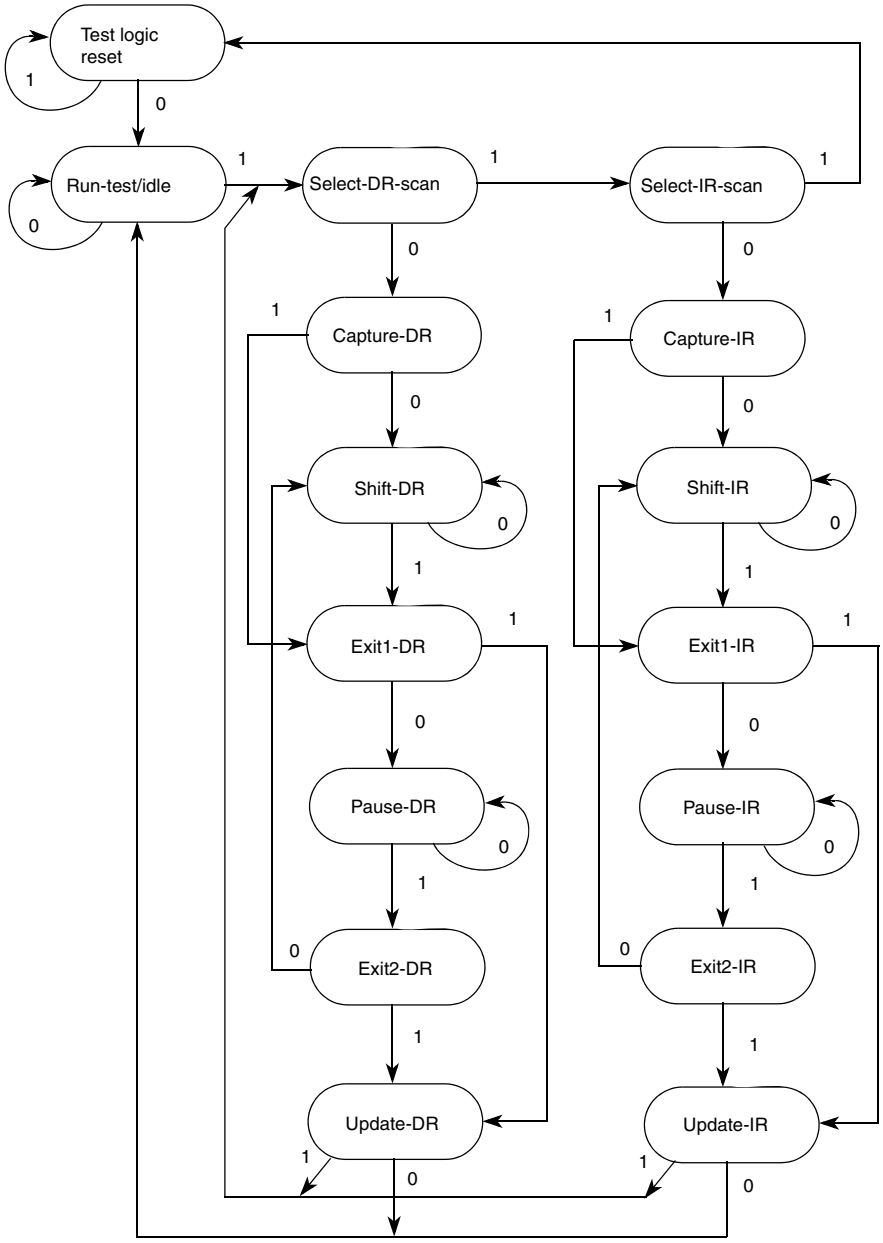


Figure 425. Shifting data through a register

32.8.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 426* 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 426* 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 426. 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 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 can be terminated after fetching the required number of bits.

32.8.4 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 414*.

Table 414. 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
ACCESS_AUX_TAP_TCU	11011	Grants the TCU ownership of the TAP
ACCESS_AUX_TAP_ONCE	10001	Grants the PLATFORM ownership of the TAP
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
Reserved	10010	—
BYPASS	11111	Selects bypass register for data operations
Factory Debug Reserved(1)	00101 00110 01010	Intended for factory debug only
Reserved(2)	All other codes	Decoded to select bypass register

1. Intended for factory debug, and not customer use
2. STMicroelectronics reserves the right to change the decoding of reserved instruction codes

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.

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.

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.

During the SAMPLE instruction, the following pad status is enforced:

- Weak pull is disabled (independent from PCRx[WPE])
- Analog switch is disabled (independent of PCRx[APC])
- Slew rate control is forced to the slowest configuration (independent from PCRx[SRC[1]])

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 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 instruction. System operation is not affected.

During the SAMPLE/PRELOAD instruction, the following pad status is enforced:

- Weak pull is disabled (independent from PCRx[WPE])
- Analog switch is disabled (independent of PCRx[APC])
- Slew rate control is forced to the slowest configuration (independent from PCRx[SRC[1]])

32.8.5 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.

32.9 e200z0 OnCE controller

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug features, as well as providing access to the Nexus2+ configuration registers. A complete discussion of the e200z0 OnCE debug features is available in the *e200z0 Reference Manual*.

32.9.1 e200z0 OnCE Controller Block Diagram

Figure 427 is a block diagram of the e200z0 OnCE block.

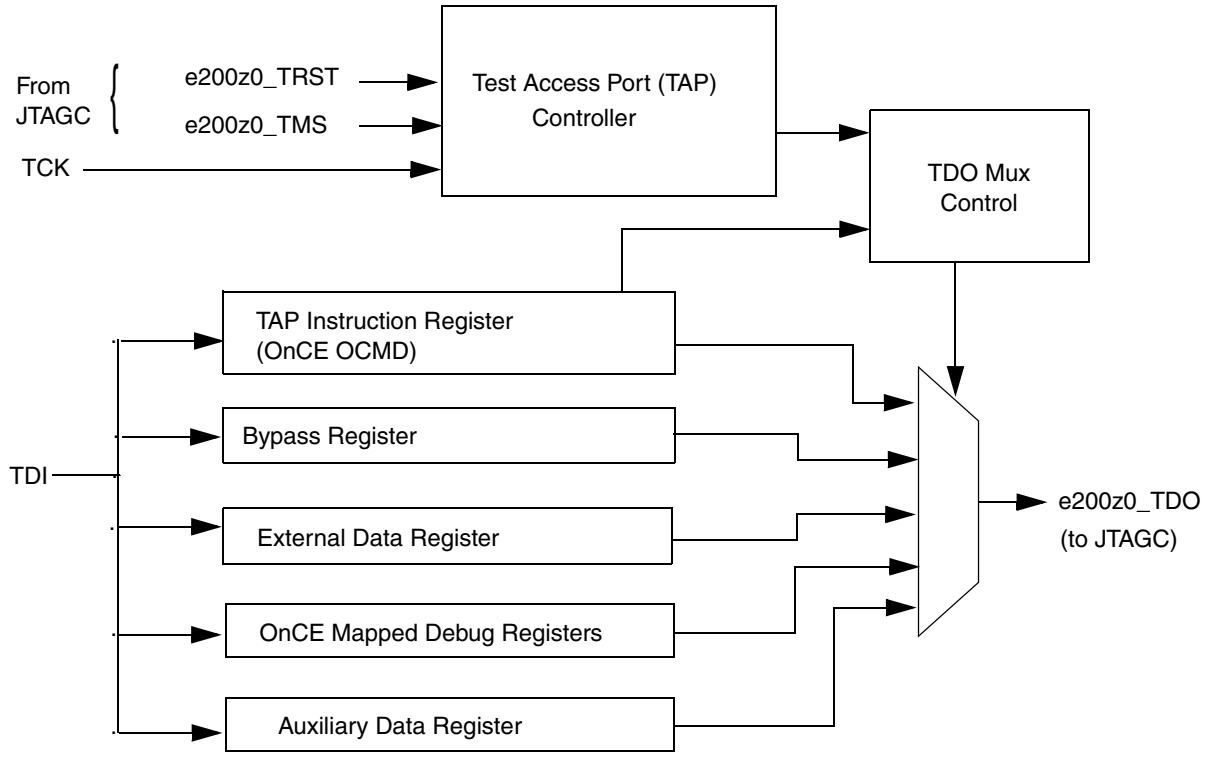


Figure 427. e200z0 OnCE Block Diagram

32.9.2 e200z0 OnCE Controller Functional Description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described below.

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*.

32.9.3 e200z0 OnCE Controller Register Description

Most e200z0 OnCE debug registers are fully documented in the *e200z0 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 *Table 428*. 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 428. OnCE Command Register (OCMD)

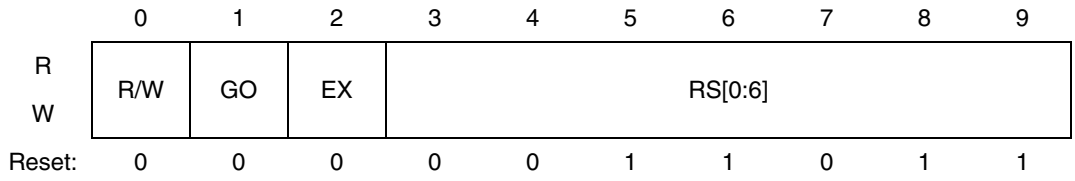


Table 415. e200z0 OnCE Register Addressing

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)
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	Nexus2+ Access

Table 415. e200z0 OnCE Register Addressing (continued)

RS[0:6]	Register Selected
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

32.10 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
2. Load the appropriate instruction for the test or action to be performed.

33 Nexus Development Interface (NDI)

33.1 Introduction

The Nexus Development Interface (NDI) block provides real-time development support capabilities for the SPC560Bx and SPC560Cx 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 SPC560Bx and SPC560Cx.

The NDI block interfaces to the e200z0, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes program trace, watchpoint messaging, ownership trace, watchpoint triggering, processor overrun control, run-time access to the MCU's internal memory map, and access to the e200z0 internal registers during halt, via the JTAG port.

33.2 Block diagram

Figure 429 shows a functional block diagram of the NDI.

A simplified block diagram of the NDI illustrates the functionality and interdependence of major blocks (see *Figure 430*) and how the individual Nexus blocks are combined to form the NDI.

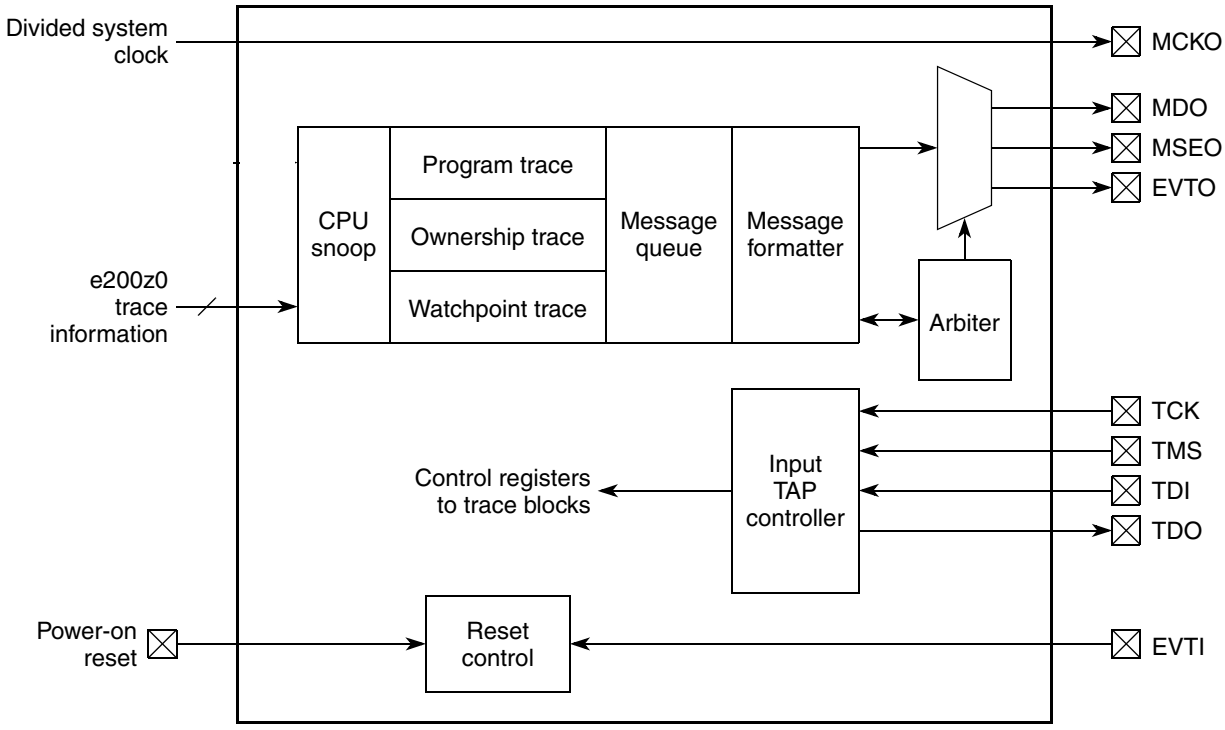


Figure 429. NDI Functional Block Diagram

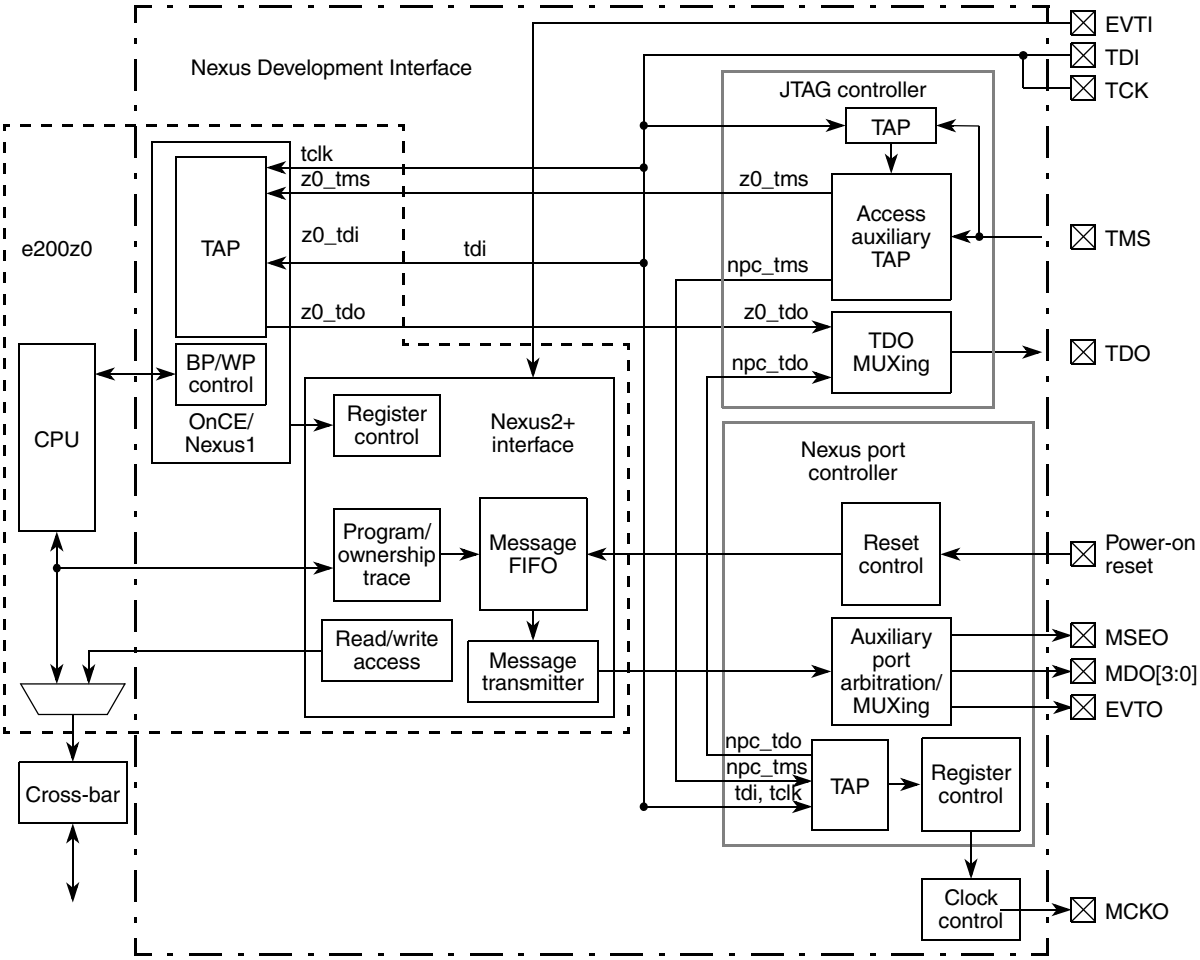


Figure 430. NDI Implementation Block Diagram

33.3 Features

The NDI module of the SPC560Bx and SPC560Cx 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.

- Watchpoint messaging via the auxiliary pins
- Watchpoint trigger enable of program trace messaging
- Auxiliary interface for higher data input/output
  - 4 message data out pins
  - 1 message start/end out pins (MSEO)
  - 1 watchpoint event pin (EVTO)
  - 1 event-in pin (EVTI)
  - 1 message clock out pin (MCKO)
  - 4-pin JTAG port (TDI, TDO, TMS, and TCK)
- Registers for program trace, ownership trace, and watchpoint trigger.
- All features controllable and configurable via the JTAG port.
- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This allows for enhanced download/upload capabilities.
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port.
- The NDI block reset is controlled with power-on reset, and the TAP state machine. All these sources are independent of system reset.
- Support for internal censorship mode to prevent external access to flash memory contents when censorship is enabled.

Note: If the e200z0 cores has executed a wait instruction, then the Nexus2+ controller clocks are gated off. While the core is in this state, it is not be possible to perform Nexus read/write operations.

33.4 Modes of Operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal or through state machine transitions controlled by TMS. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAGC controller (JTAGC) block.

The Nexus port controller (NPC) transitions out of the reset state immediately following negation of power-on reset.

33.4.1 Nexus Reset

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to three-state the output pins or use them for another function.

33.4.2 Operating Mode

In full-port mode, all available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. Four MDO pins are available in full-port mode.

Disabled-Port Mode

In disabled-port mode, message transmission is disabled. Any debug feature that generates messages can not be used. The primary features available are class 1 features and read/write access.

Censored Mode

The NDI supports internal flash censorship mode by preventing the transmission of trace messages and Nexus access to memory-mapped resources when censorship is enabled.

Stop Mode

Stop mode logic is implemented in the NPC. When a request is made to enter STOP mode, the NDI block completes monitoring of any pending bus transaction, transmits all messages already queued, and acknowledges the stop request. After the acknowledgment, the system clock input are shut off by the clock driver on the device. While the clocks are shut off, the development tool cannot access NDI registers via the JTAG port.

33.5 External Signal Description

All the signals are available in the 208BGA without any multiplexing scheme. Refer to *Chapter 4 Signal description* for details.

33.5.1 Nexus Signal Reset States

Table 416. NDI Signal Reset State

Name	Function	Nexus Reset State	Pull
EVTI	Event-in pin	—	Up
EVTO	Event-out pin	0b1	—
MCKO	Message clock out pin	0b0	—
MDO[3:0]	Message data out pins	0	—
MSEO	Message start/end out pin	0b1	—

33.6 Memory Map and Register Description

The NDI block contains no memory-mapped registers. Nexus registers are accessed by a development tool via the JTAG port using a client-select value and a register index. OnCE registers are accessed by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD) via the JTAG port.

33.6.1 Nexus Debug Interface Registers

Table 417 shows the NDI registers by client select and index values. OnCE register addressing is documented in the JTAGC chapter of this reference manual.

Table 417. Nexus Debug Interface Registers

Client select	Index	Register	Location
Client-Independent Registers			
0bxxxx	0	Nexus Device ID (DID) Register(1)	*on page 33-785*
0bxxxx	127	Port Configuration Register (PCR)(1)	*on page 33-786*
e200z0 Control/Status Registers			
0b0000	2	Development Control Register 1 (DC1)	*on page 33-788*
0b0000	3	Development Control Register 2 (DC2)	*on page 33-788*
0b0000	4	Development Status (DS) Register	*on page 33-790*
0b0000	7	Read/Write Access Control/Status (RWCS) Register	*on page 33-792*
0b0000	9	Read/Write Access Address (RWA) Register	*on page 33-793*
0b0000	10	Read/Write Access Data (RWD) Register	*on page 33-794*
0b0000	11	Watchpoint Trigger (WT) Register	*on page 33-794*

1. Implemented in NPC block. All other registers implemented in e200z0 Nexus2+ block.

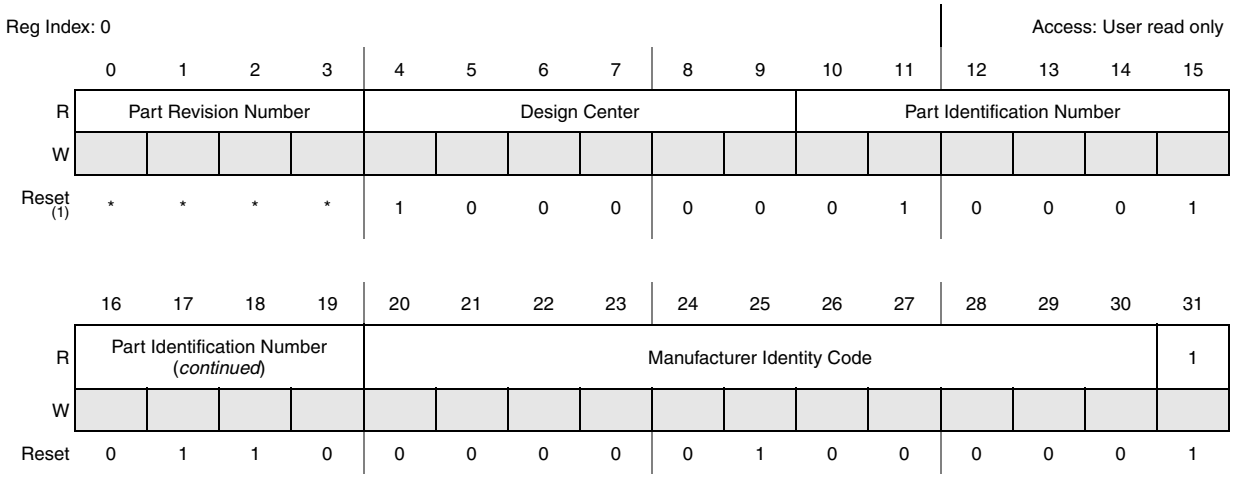
33.6.2 Register Description

This section lists the NDI registers and describes the registers and their bit fields.

Nexus Device ID (DID) Register

The NPC device identification register, shown in *Figure 431*, allows the part revision number, design center, part identification number, and manufacturer identity code of the device to be determined through the auxiliary output port, and serially through TDO. This register is read-only.

Figure 431. Nexus Device ID (DID) Register



1. Part Revision Number default value is 0x0 for the device's initial mask set and changes for each mask set revision.

Table 418. DID field descriptions

Field	Description
0–3 PRN	Part Revision Number Contains the revision number of the part. This field changes with each revision of the device or module.
4–9 DC	Design Center
10–19 PIN	Part Identification Number Contains the part number of the device.
20–30 MIC	Manufacturer Identity Code Contains the reduced Joint Electron Device Engineering Council (JEDEC) IDfor STMicroelectronics, 0x20.
31	Fixed per JTAG 1149.1 Always set to 1.

Port Configuration Register (PCR)

The PCR 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 NDI 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_EN, SLEEP_SYNC, and STOP_SYNC 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.

Figure 432. Port Configuration Register (PCR)

Reg Index: 127 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	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	LP_DBG_EN	0	0	0	0	0	SLEEP_SYNC	STOP_SYNC	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

Table 419. PCR field descriptions

Field	Description
0 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. 0 A subset of MDO pins are used to transmit messages. 1 All MDO pins are used to transmit messages.
1 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. 0 MCKO gating is disabled. 1 MCKO gating is enabled.
2 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. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.

Table 419. PCR field descriptions (continued)

Field	Description
3–5 MCKO_DIV [2:0]	MCKO Division Factor The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. SYS_CLK represents the system clock frequency: **Table 0-1** **Value** **MCKO frequency** 0b000 SYS_CLK 0b001 SYS_CLK÷2 (default value if a reserved encoding is programmed) 0b010 Reserved 0b011 SYS_CLK÷4 *Note: MCKO_DIV value and associated MCKO frequency should be configured taking into account the frequency limitation of the associated MCKO pad. Please refer to datasheet IO section.*
6 EVT_EN	EVTO/EVTI Enable This bit enables the EVTO/EVTI port functions. 0 EVTO/EVTI port disabled 1 EVTO/EVTI port enabled
7–15	Reserved
16 LP_DBG_EN	Low Power Debug Enable The LP_DBG_EN bit enables debug functionality to support entry and exit from low power sleep and STOP modes. 0 Low power debug disabled 1 Low power debug enabled
17–21	Reserved
22 SLEEP_SYNC	Sleep Mode Synchronization The SLEEP_SYNC bit is used to synchronize the entry into sleep mode between the device and debug tool. The device sets this bit before a pending entry into sleep mode. After reading SLEEP_SYNC as set, the debug tool then clears SLEEP_SYNC to acknowledge to the device that it may enter into sleep mode. 0 Sleep mode entry acknowledge 1 Sleep mode entry pending
23 STOP_SYNC	Stop Mode Synchronization The STOP_SYNC bit is used to synchronize the entry into STOP mode between the device and debug tool. The device sets this bit before a pending entry into STOP mode. After reading STOP_SYNC as set, the debug tool then clears STOP_SYNC to acknowledge to the device that it may enter into STOP mode. 0 Stop mode entry acknowledge 1 Stop mode entry pending
24–30	Reserved
31 PSTAT_EN	Processor Status Mode Enable

Development Control Register 1, 2 (DC1, DC2)

The development control registers are used to control the basic development features of the Nexus module. *Figure 433* shows development control register 1 and *Table 420* describes the register's fields.

Figure 433. Development Control Register 1 (DC1)

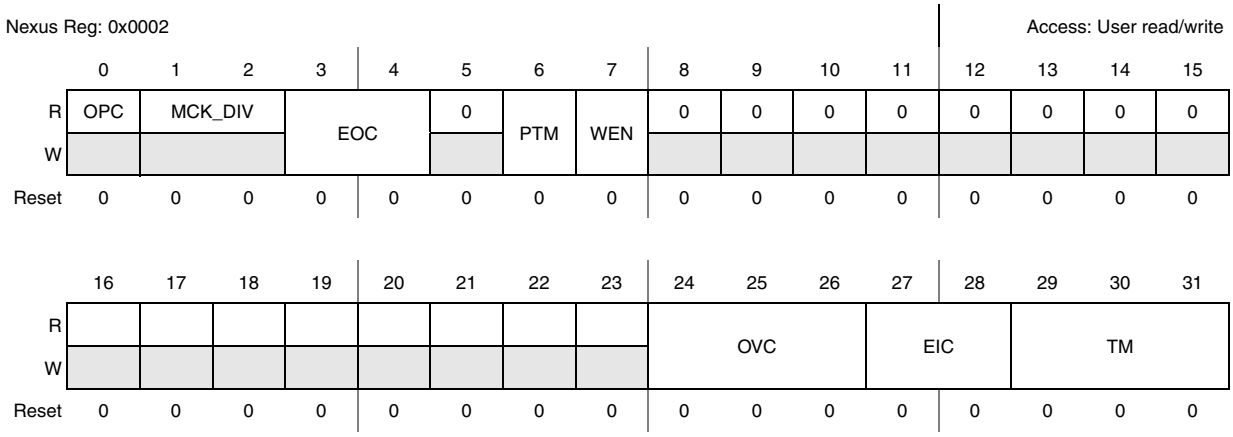


Table 420. DC1 field descriptions

Field	Description
0 OPC(1)	Output Port Mode Control 0 Reduced-port mode configuration (2 MDO pins) 1 Full-port mode configuration (4 MDO pins)
1–2 MCK_DIV[1:0](1)	MCKO Clock Divide Ratio (see note below) 00 MCKO is 1x processor clock freq. 01 MCKO is 1/2x processor clock freq. 10 MCKO is 1/4x processor clock freq. 11 MCKO is 1/8x processor clock freq.
3–4 EOC[1:0]	EVTO Control 00 EVTO upon occurrence of watchpoints (configured in DC2) 01 EVTO upon entry into debug mode 10 EVTO upon timestamping event 11 Reserved
5	Reserved
6 PTM	Program Trace Method 0 Program trace uses traditional branch messages. 1 Program trace uses branch history messages.
7 WEN	Watchpoint Trace Enable 0 Watchpoint messaging disabled 1 Watchpoint messaging enabled
8–23	Reserved

Table 420. DC1 field descriptions (continued)

Field	Description
24–26 OVC[2:0]	Overrun Control 000 Generate overrun messages. 001–010 Reserved 011 Delay processor for BTM / DTM / OTM overruns. 1XX Reserved
27–28 EIC[1:0]	EVTI Control 00 EVTI is used for synchronization (program trace/ data trace) 01 EVTI is used for debug request 1X Reserved
29–31 TM[2:0]	Trace Mode Any or all of the TM bits may set, enabling one or more traces. 000 No trace 1XX Program trace enabled X1X Data trace enabled (not supported mode) XX1 Ownership trace enabled

1. The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.

Development control register 2 is shown in *Figure 434* and its fields are described in *Table 421*.

Figure 434. Development Control Register 2 (DC2)

Nexus Reg: 0x0003 — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EWC								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 421. DC2 field descriptions

Field	Description
0–7 EWC[7:0]	EVTO Watchpoint Configuration Any or all of the bits in EWC may be set to configure the EVTO watchpoint. 00000000No Watchpoints trigger EVTO 1XXXXXXXWatchpoint #0 (IAC1 from Nexus1) triggers EVTO. X1XXXXXXWatchpoint #1 (IAC2 from Nexus1) triggers EVTO. XX1XXXXXWatchpoint #2 (IAC3 from Nexus1) triggers EVTO. XXX1XXXXWatchpoint #3 (IAC4 from Nexus1) triggers EVTO. XXXX1XXXWatchpoint #4 (DAC1 from Nexus1) triggers EVTO. XXXXX1XXWatchpoint #5 (DAC2 from Nexus1) triggers EVTO. XXXXXX1XWatchpoint #6 (DCNT1 from Nexus1) triggers EVTO. XXXXXXX1Watchpoint #7 (DCNT2 from Nexus1) triggers EVTO.
8–31	Reserved

Note: The EOC bits in DC1 must be programmed to trigger EVTO on watchpoint occurrence for the EWC bits to have any effect.

Development Status (DS) Register

The development status register is used to report system debug status. When debug mode is entered or exited, or a core-defined low-power mode is entered, a debug status message is transmitted with DS[31:24]. The external tool can read this register at any time.

Figure 435. Development Status (DS) Register

Nexus Reg: 0x0004 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DBG	0	0	0	LPC		CHK	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 422. DS field descriptions

Field	Description
0 DBG	CPU Debug Mode Status 0 CPU not in debug mode 1 CPU in debug mode
1–3	Reserved

Table 422. DS field descriptions (continued)

Field	Description
4–5 LPC[1:0]	CPU Low-Power Mode Status 00 Normal (run) mode 01 CPU in halted state 10 CPU in stopped state 11 Reserved
6 CHK	CPU Checkstop Status 0 CPU not in checkstop state 1 CPU in checkstop state
7–31	Reserved

Read/Write Access Control/Status (RWCS) Register

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus while the processor is halted or during runtime. The RWCS register also provides read/write access status information as shown in *Table 424*.

Figure 436. Read/Write Access Control/Status (RWCS) Register

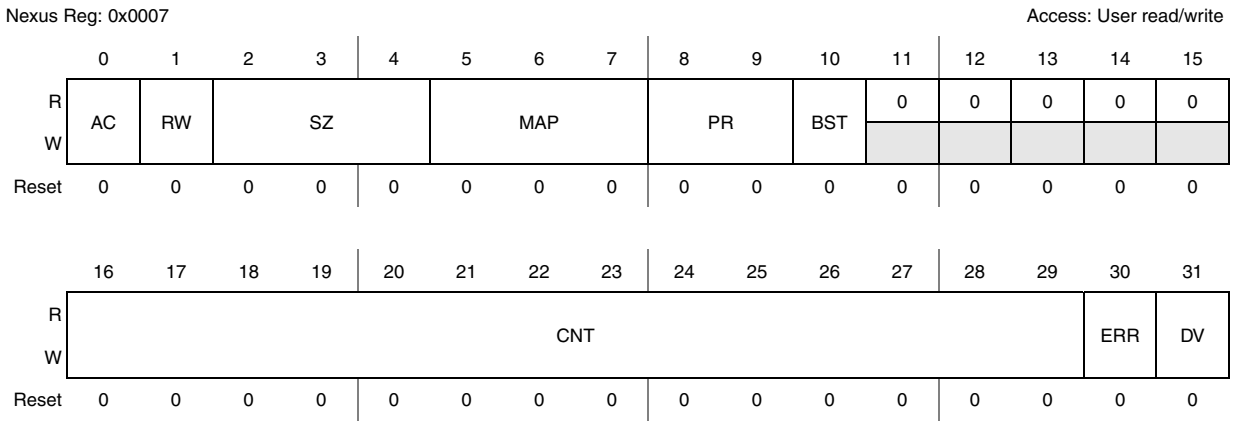


Table 423. RWCS field descriptions

Field	Description
0 AC	Access Control 0 End access. 1 Start access.
1 RW	Read/Write Select 0 Read access 1 Write access
2–4 SZ[2:0]	Word Size 000 8-bit (byte) 001 16-bit (halfword) 010 32-bit (word) 011 64-bit (doubleword—only in burst mode) 100–111 Reserved (default to word)
5–7 MAP[2:0]	MAP Select 000 Primary memory map 001–111 Reserved
8–9 PR[1:0]	Read/Write Access Priority 00 Lowest access priority 01 Reserved (default to lowest priority) 10 Reserved (default to lowest priority) 11 Highest access priority
10 BST	Burst Control 0 Module accesses are single bus cycle at a time. 1 Module accesses are performed as burst operation.
11–15	Reserved

Table 423. RWCS field descriptions (continued)

Field	Description
16–31 CNT[13:0]	Access Control Count Number of accesses of word size SZ
30 ERR	Read/Write Access Error See *Table 424*.
31 DV	Read/Write Access Data Valid See *Table 424*.

Table 424 details the status bit encodings.

Table 424. Read/Write Access Status Bit Encoding

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

Read/Write Access Address (RWA) Register

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

Figure 437. Read/Write Access Address (RWA) Register

Nexus Reg: 0x0009 Access: User read/write

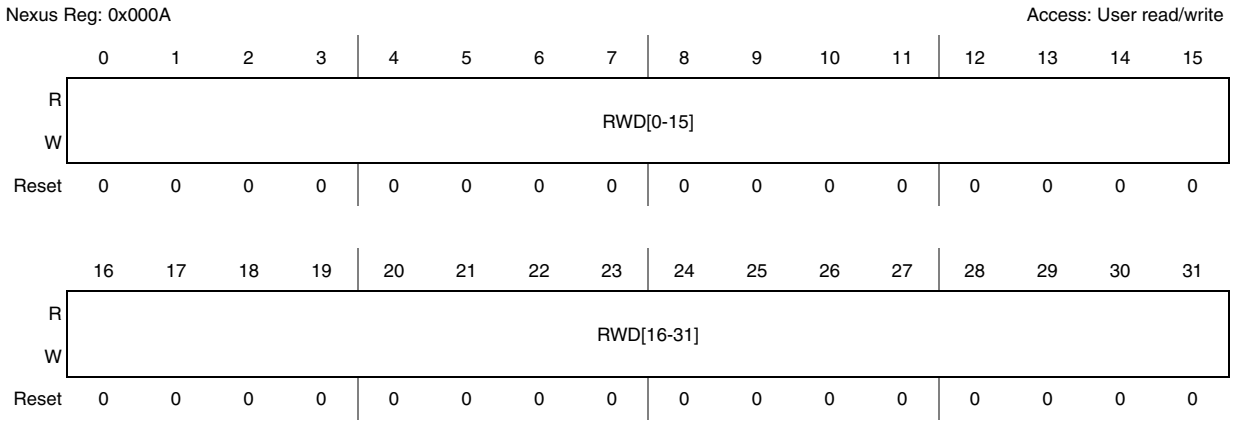
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	RWA[0-15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Read/Write Access Data (RWD) Register

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

Figure 438. Read/Write Access Data (RWD) Register



Watchpoint Trigger (WT) Register

The watchpoint trigger register allows the watchpoints defined within the Nexus1 logic to trigger actions. These watchpoints can control program and/or data trace enable and disable. The WT bits can be used to produce an address-related window for triggering trace messages.

Figure 439. Watchpoint Trigger (WT) Register

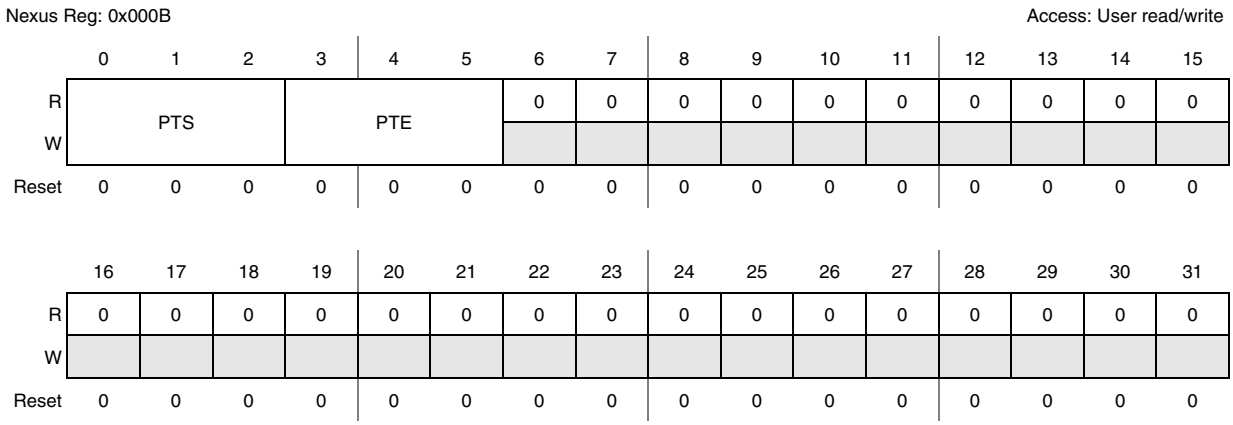


Table 425 details the watchpoint trigger register fields.

Table 425. WT field descriptions

Field	Description
0–2 PTS[2:0]	Program Trace Start Control 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
3–5 PTE[2:0]	Program Trace End Control 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
12–31	Reserved

33.7 Functional description

The NDI block is implemented by integrating the following blocks on the SPC560Bx and SPC560Cx:

- Nexus e200z0 development interface (OnCE and Nexus2p subblocks)
- Nexus port controller (NPC) block
- NPC_HNDSHK module

33.7.1 NPC_HNDSHK module

This module enables debug entry/exit across low power modes (Stop, Halt, standby).

The NPC_HNDSHK supports:

- Setting and clearing of the NPC PCR sync bit on low-power mode entry and exit
- Putting the core into debug mode on low-power mode exit
- Generating a falling edge on the JTAG TDO pad on low-power mode exit

On HALT, STOP, or STANDBY mode entry, the MC_ME asserts the lp_mode_entry_req input after the clock disable process has completed and before the processor enters its halted or stopped state. The mode transition will then not proceed until the lp_mode_entry_ack output has been asserted. The notification to the debugger of a low-power mode entry consists of setting the low-power mode handshake bit in the port control register (read by the debugger) via the lp_sync_in output. The debugger acknowledges that the transition into a low-power mode may proceed by clearing the low-power mode handshake bit in the port control register (written by the debugger), which results in the deassertion of the lp_sync_out input.

In anticipation of the low-power mode exit notification, the TDO pad is driven to `1'.

On HALT or STOP mode exit, the MC_ME asserts the lp_mode_exit_req input after ensuring that the regulator and memories are in normal mode and before the processor exits its halted or stopped state. The mode transition will then not proceed until the lp_mode_exit_ack output has been asserted. The MC_RGM asserts the exit_from_standby input when executing a reset sequence due to a STANDBY exit. The reset sequence will then not complete until the lp_mode_exit_ack output has been asserted.

The notification to the debugger of a low-power mode exit consists of driving the TDO pad to `0'. The debugger acknowledges that the transition from a low-power mode can continue by setting the low-power mode sync bit in the port control register (written by debugger), which results in the assertion of the lp_sync_out input.

Note: The debugger clock multiplexer may not guarantee glitch free switching. Therefore, TCK should be disabled from when the debugger clears the sync bit in ENTRY_CLR until the debugger senses the falling edge of TDO in TDO_SET.

33.7.2 Enabling Nexus Clients for TAP Access

After the conditions have been met to bring the NDI out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in *Table 426*. After the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. The NEXUS-ENABLE instruction opcode for each Nexus client is listed in *Table 427*. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

Table 426. JTAGC Instruction Opcodes to Enable Nexus Clients

JTAGC Instruction	Opcode	Description
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z0 TAP controller

Table 427. Nexus Client JTAG Instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus ENABLE instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z0 OnCE JTAG Instruction Opcodes(1)		
NEXUS2_ACCESS	Opcode for e200z0 OnCE Nexus ENABLE instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z0 OnCE BYPASS instruction (10-bits)	0x7F

1. Refer to the e200z0 reference manual for a complete list of available OnCE instructions.

33.7.3 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC

is then required to enable the NDI and select the mode of operation. Asserting MCKO_EN in the PCR places the NDI 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. When writing to the PCR, the PCR LSB must be written to a logic zero. Setting the LSB of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 428 describes the NDI configuration options.

Table 428. NDI configuration options

MCKO_EN bit of PCR	FPM bit of PCR	Configuration
0	X	Disabled
1	1	Full-Port Mode
1	0	Reduced Port Mode

33.7.4 Programmable MCKO Frequency

MCKO is an output clock to the development tools used for the timing of 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 port configuration register (PCR) located in the NPC. Possible operating frequencies include one-quarter and one-eighth system clock speed.

Refer to the MCKO_DIV [2:0] field description in *Table 419* for the MCKO_DIV encodings, where SYS_CLK represents the system clock frequency. The default value selected if a reserved encoding is programmed is SYS_CLK÷2.

33.7.5 Nexus Messaging

Most of the messages transmitted by the NDI include an SRC field. This field is used to identify which source generated the message. *Table 429* shows the values used for the SRC field by the different clients on the SPC560Bx and SPC560Cx. These values are specific to the SPC560Bx and SPC560Cx. The size of the SRC field in transmitted messages is 4 bits. This value is also specific to the SPC560Bx and SPC560Cx.

Table 429. SRC Packet Encodings

SRC[3:0]	SPC560Bx and SPC560Cx Client
0b0000	e200z0
All other combinations	Reserved

33.7.6 EVTO Sharing

The NPC block controls sharing of the EVTO output between all Nexus clients that generate an EVTO signal. The sharing mechanism is a logical AND of all incoming EVTO signals from Nexus blocks, thereby asserting EVTO whenever any block drives its EVTO. When there is no active MCKO, such as in disabled mode, the NPC drives EVTO for two system clock periods. EVTO sharing is active as long as the NDI is not in reset.

33.7.7 Debug Mode Control

On SPC560Bx and SPC560Cx, program breaks can be requested either by using the EVTI pin as a break request, or when a Nexus event is triggered.

EVTI Generated Break Request

To use the EVTI pin as a debug request, the EIC field in the e200z0 Nexus2+ Development Control Register 1 (DC1[4:3]) must be set to configure the EVTI input as a debug request.

33.7.8 Ownership Trace

Overview

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

Ownership Trace Messaging (OTM)

Ownership trace information is messaged via the auxiliary port using an ownership trace message (OTM). The e200z0h processor contains a Power Architecture platform defined process ID register within the CPU.

The process ID register is updated by the operating system software to provide task/process ID information. The contents of this register are replicated on the pins of the processor and connected to Nexus. The process ID register value can be accessed using the **mfspr**/**mtspr** instructions.

There is one condition which will cause an ownership trace message: When new information is updated in the OTR register or process ID register by the e200z0h processor, the data is latched within Nexus, and is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

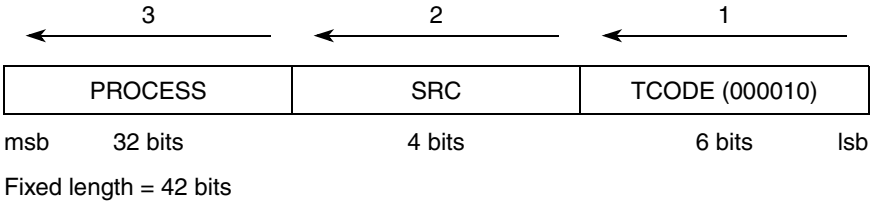


Figure 440. Ownership Trace Message Format

OTM Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard incoming messages until it has completely emptied the queue. Once emptied, an error message will be queued. The error encoding will indicate which types of messages attempted to be queued while the FIFO was being emptied.

If only an OTM message attempts to enter the queue while it is being emptied, the error message will incorporate the OTM only error encoding (00000). If both OTM and either BTM or DTM messages attempt to enter the queue, the error message will incorporate the OTM

and (program or data) trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message will incorporate error encoding (01000).

Note: The OVC bits within the DC1 register can be set to delay the CPU in order to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see *Table 430*)

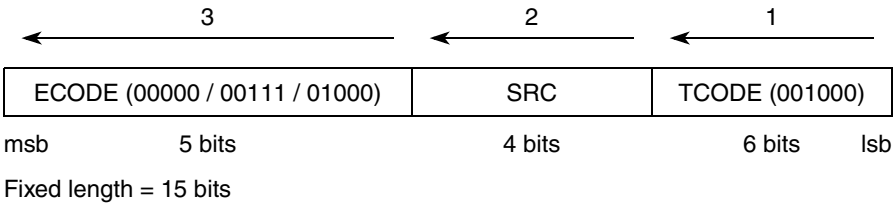


Figure 441. Error Message Format

Table 430. Error Code Encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Ownership trace overrun
00001	Program trace overrun
00010	Data trace overrun
00011	Read/write access error
00101	Invalid access opcode (Nexus register unimplemented)
00110	Watchpoint overrun
00111	(Program trace or data trace) and ownership trace overrun
01000	(Program trace or data trace or ownership trace) and watchpoint overrun
01001–0111	Reserved
11000	BTM lost due to collision w/ higher priority message
11001–11111	Reserved

OTM Flow

Ownership trace messages are generated when the operating system writes to the e200z0h process ID register or the memory mapped ownership trace register.

The following flow describes the OTM process:

1. The process ID register is a system control register. It is internal to the e200z0h processor and can be accessed by using PPC instructions **mtspr** and **mfspr**. The

contents of this register are replicated on the pins of the processor and connected to Nexus.

2. OTR/process ID register reads do not cause ownership trace messages to be transmitted by the NZ0H module.
3. If the periodic OTM message counter expires (after 255 queued messages without an OTM), an OTM is sent using the latched data from the previous OTM or process ID register write.

Appendix A Register Map

Table 431. Module base addresses

Module name	Base addresses	Page
Code Flash A Configuration	0xC3F8_8000	on page A-802
Data Flash A Configuration	0xC3F8_C000	on page A-802
System Integration Unit Lite (SIUL)	0xC3F9_0000	on page A-803
WakeUp Unit	0xC3F9_4000	on page A-803
eMIOS_0	0xC3FA_0000	on page A-811
eMIOS_1	0xC3FA_4000	on page A-816
System Status and Configuration Module (SSCM)	0xC3FD_8000	on page A-822
Mode Entry Module (MC_ME)	0xC3FD_C000	on page A-822
FXOSC	0xC3FE_0000	on page A-825
SXOSC	0xC3FE_0040	on page A-825
FIRC	0xC3FE_0060	on page A-825
SIRC	0xC3FE_0080	on page A-825
FMPLL	0xC3FE_00A0	on page A-825
CMU	0xC3FE_0100	on page A-825
Clock Generation Module (MC_CGM)	0xC3FE_0370	on page A-825
Reset Generation Module (MC_RGM)	0xC3FE_4000	on page A-826
Power Control Unit (MC_PCU)	0xC3FE_8000	on page A-826
Real Time Counter (RTC/API)	0xC3FE_C000	on page A-826
Periodic Interrupt Timer (PIT)	0xC3FF_0000	on page A-827
ADC	0xFFE0_0000	on page A-827
I2C	0xFFE3_0000	on page A-831
LINFlex_0	0xFFE4_0000	on page A-831
LINFlex_1	0xFFE4_4000	on page A-832
LINFlex_2	0xFFE4_8000	on page A-833
LINFlex_3	0xFFE4_C000	on page A-833
CTU	0xFFE6_4000	on page A-834
CAN sampler	0xFFE7_0000	on page A-836
MPU	0xFFF1_0000	on page A-836
SWT	0xFFF3_8000	on page A-837
STM	0xFFF3_C000	on page A-837
ECSM	0xFFF4_0000	on page A-838
INTC	0xFFF4_8000	on page A-839

Table 431. Module base addresses (continued)

Module name	Base addresses	Page
DSPI_0	0xFFF9_0000	*on page A-841*
DSPI_1	0xFFF9_4000	*on page A-842*
DSPI_2	0xFFF9_8000	*on page A-843*
FlexCAN_0 (CAN0)	0xFFFC_0000	*on page A-844*
FlexCAN_1 (CAN1)	0xFFFC_4000	*on page A-850*
FlexCAN_2 (CAN2)	0xFFFC_8000	*on page A-855*
FlexCAN_3 (CAN3)	0xFFFC_C000	*on page A-861*
FlexCAN_4 (CAN4)	0xFFFD_0000	*on page A-866*
FlexCAN_5 (CAN5)	0xFFFD_4000	*on page A-872*

Table 432. Detailed register map

Register description	Register name	Used size	Address
Code Flash A Configuration			**0xC3F8_8000**
Module Configuration Register	CFLASH_MCR	32-bit	Base + 0x0000
Low/Mid Address Space Block Locking Register	CFLASH_LML	32-bit	Base + 0x0004
High Address Space Block Locking Register	CFLASH_HBL	32-bit	Base + 0x0008
Secondary Low/Mid Address Space Block Locking Register	CFLASH_SLL	32-bit	Base + 0x000C
Low/Mid Address Space Block Select Register	CFLASH_LMS	32-bit	Base + 0x0010
High Address Space Block Select Register	CFLASH_HBS	32-bit	Base + 0x0014
Address Register	CFLASH_ADR	32-bit	Base + 0x0018
Bus Interface Unit Register 0	CFLASH_BIU0	32-bit	Base + 0x001C
Bus Interface Unit Register 1	CFLASH_BIU1	32-bit	Base + 0x0020
Bus Interface Unit Register 2	CFLASH_BIU2	32-bit	Base + 0x0024
Reserved	—	—	(Base + 0x0028) – (Base + 0x003B)
User Test Register 0	CFLASH_UT0	32-bit	Base + 0x003C
User Test Register 1	CFLASH_UT1	32-bit	Base + 0x0040
User Test Register 2	CFLASH_UT2	32-bit	Base + 0x0044
User Multiple Input Signature Register 0	CFLASH_UMISR0	32-bit	Base + 0x0048
User Multiple Input Signature Register 1	CFLASH_UMISR1	32-bit	Base + 0x004C
User Multiple Input Signature Register 2	CFLASH_UMISR2	32-bit	Base + 0x0050
User Multiple Input Signature Register 3	CFLASH_UMISR3	32-bit	Base + 0x0054
User Multiple Input Signature Register 4	CFLASH_UMISR4	32-bit	Base + 0x0058
Data Flash A Configuration			**0xC3F8_C000**

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Module Configuration Register	DFLASH_MCR	32-bit	Base + 0x0000
Low/Mid Address Space Block Locking Register	DFLASH_LML	32-bit	Base + 0x0004
High Address Space Block Locking Register	DFLASH_HBL	32-bit	Base + 0x0008
Secondary Low/Mid Address Space Block Locking Register	DFLASH_SLL	32-bit	Base + 0x000C
Low/Mid Address Space Block Select Register	DFLASH_LMS	32-bit	Base + 0x0010
High Address Space Block Select Register	DFLASH_HBS	32-bit	Base + 0x0014
Address Register	DFLASH_ADR	32-bit	Base + 0x0018
Reserved	—	—	(Base + 0x001C) – (Base + 0x003B)
User Test Register 0	DFLASH_UT0	32-bit	Base + 0x003C
User Test Register 1	DFLASH_UT1	32-bit	Base + 0x0040
User Test Register 2	DFLASH_UT2	32-bit	Base + 0x0044
User Multiple Input Signature Register 0	DFLASH_UMISR0	32-bit	Base + 0x0048
User Multiple Input Signature Register 1	DFLASH_UMISR1	32-bit	Base + 0x004C
User Multiple Input Signature Register 2	DFLASH_UMISR2	32-bit	Base + 0x0050
User Multiple Input Signature Register 3	DFLASH_UMISR3	32-bit	Base + 0x0054
User Multiple Input Signature Register 4	DFLASH_UMISR4	32-bit	Base + 0x0058
System Integration Unit Lite (SIUL)			**0xC3F9_0000**
Reserved	—	—	Base + (0x0000 – 0x0003)
MCU ID Register 1	MIDR1	32-bit	Base + 0x0004
MCU ID Register 2	MIDR2	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x0013)
Interrupt Status Flag Register	ISR	32-bit	Base + 0x0014
Interrupt Request Enable Register	IRER	32-bit	Base + 0x0018
Reserved	—	—	Base + (0x001C – 0x0027)
Interrupt Rising Edge Event Enable	IREER	32-bit	Base + 0x0028
Interrupt Falling-Edge Event Enable	IFEER	32-bit	Base + 0x002C
IFER Interrupt Filter Enable Register	IFER	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034 – 0x003F)
Pad Configuration Register 0	PCR0	16-bit	Base + 0x0040
Pad Configuration Register 1	PCR1	16-bit	Base + 0x0042
Pad Configuration Register 2	PCR2	16-bit	Base + 0x0044

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Pad Configuration Register 3	PCR3	16-bit	Base + 0x0046
Pad Configuration Register 4	PCR4	16-bit	Base + 0x0048
Pad Configuration Register 5	PCR5	16-bit	Base + 0x004A
Pad Configuration Register 6	PCR6	16-bit	Base + 0x004C
Pad Configuration Register 7	PCR7	16-bit	Base + 0x004E
Pad Configuration Register 8	PCR8	16-bit	Base + 0x0050
Pad Configuration Register 9	PCR9	16-bit	Base + 0x0052
Pad Configuration Register 10	PCR10	16-bit	Base + 0x0054
Pad Configuration Register 11	PCR11	16-bit	Base + 0x0056
Pad Configuration Register 12	PCR12	16-bit	Base + 0x0058
Pad Configuration Register 13	PCR13	16-bit	Base + 0x005A
Pad Configuration Register 14	PCR14	16-bit	Base + 0x005C
Pad Configuration Register 15	PCR15	16-bit	Base + 0x005E
Pad Configuration Register 16	PCR16	16-bit	Base + 0x0060
Pad Configuration Register 17	PCR17	16-bit	Base + 0x0062
Pad Configuration Register 18	PCR18	16-bit	Base + 0x0064
Pad Configuration Register 19	PCR19	16-bit	Base + 0x0066
Pad Configuration Register 20	PCR20	16-bit	Base + 0x0068
Pad Configuration Register 21	PCR21	16-bit	Base + 0x006A
Pad Configuration Register 22	PCR22	16-bit	Base + 0x006C
Pad Configuration Register 23	PCR23	16-bit	Base + 0x006E
Pad Configuration Register 24	PCR24	16-bit	Base + 0x0070
Pad Configuration Register 25	PCR25	16-bit	Base + 0x0072
Pad Configuration Register 26	PCR26	16-bit	Base + 0x0074
Pad Configuration Register 27	PCR27	16-bit	Base + 0x0076
Pad Configuration Register 28	PCR28	16-bit	Base + 0x0078
Pad Configuration Register 29	PCR29	16-bit	Base + 0x007A
Pad Configuration Register 30	PCR30	16-bit	Base + 0x007C
Pad Configuration Register 31	PCR31	16-bit	Base + 0x007E
Pad Configuration Register 32	PCR32	16-bit	Base + 0x0080
Pad Configuration Register 33	PCR33	16-bit	Base + 0x0082
Pad Configuration Register 34	PCR34	16-bit	Base + 0x0084
Pad Configuration Register 35	PCR35	16-bit	Base + 0x0086
Pad Configuration Register 36	PCR36	16-bit	Base + 0x0088

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Pad Configuration Register 37	PCR37	16-bit	Base + 0x008A
Pad Configuration Register 38	PCR38	16-bit	Base + 0x008C
Pad Configuration Register 39	PCR39	16-bit	Base + 0x008E
Pad Configuration Register 40	PCR40	16-bit	Base + 0x0090
Pad Configuration Register 41	PCR41	16-bit	Base + 0x0092
Pad Configuration Register 42	PCR42	16-bit	Base + 0x0094
Pad Configuration Register 43	PCR43	16-bit	Base + 0x0096
Pad Configuration Register 44	PCR44	16-bit	Base + 0x0098
Pad Configuration Register 45	PCR45	16-bit	Base + 0x009A
Pad Configuration Register 46	PCR46	16-bit	Base + 0x009C
Pad Configuration Register 47	PCR47	16-bit	Base + 0x009E
Pad Configuration Register 48	PCR48	16-bit	Base + 0x00A0
Pad Configuration Register 49	PCR49	16-bit	Base + 0x00A2
Pad Configuration Register 50	PCR50	16-bit	Base + 0x00A4
Pad Configuration Register 51	PCR51	16-bit	Base + 0x00A6
Pad Configuration Register 52	PCR52	16-bit	Base + 0x00A8
Pad Configuration Register 53	PCR53	16-bit	Base + 0x00AA
Pad Configuration Register 54	PCR54	16-bit	Base + 0x00AC
Pad Configuration Register 55	PCR55	16-bit	Base + 0x00AE
Pad Configuration Register 56	PCR56	16-bit	Base + 0x00B0
Pad Configuration Register 57	PCR57	16-bit	Base + 0x00B2
Pad Configuration Register 58	PCR58	16-bit	Base + 0x00B4
Pad Configuration Register 59	PCR59	16-bit	Base + 0x00B6
Pad Configuration Register 60	PCR60	16-bit	Base + 0x00B8
Pad Configuration Register 61	PCR61	16-bit	Base + 0x00BA
Pad Configuration Register 62	PCR62	16-bit	Base + 0x00BC
Pad Configuration Register 63	PCR63	16-bit	Base + 0x00BE
Pad Configuration Register 64	PCR64	16-bit	Base + 0x00C0
Pad Configuration Register 65	PCR65	16-bit	Base + 0x00C2
Pad Configuration Register 66	PCR66	16-bit	Base + 0x00C4
Pad Configuration Register 67	PCR67	16-bit	Base + 0x00C6
Pad Configuration Register 68	PCR68	16-bit	Base + 0x00C8
Pad Configuration Register 69	PCR69	16-bit	Base + 0x00CA
Pad Configuration Register 70	PCR70	16-bit	Base + 0x00CC

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Pad Configuration Register 71	PCR71	16-bit	Base + 0x00CE
Pad Configuration Register 72	PCR72	16-bit	Base + 0x00D0
Pad Configuration Register 73	PCR73	16-bit	Base + 0x00D2
Pad Configuration Register 74	PCR74	16-bit	Base + 0x00D4
Pad Configuration Register 75	PCR75	16-bit	Base + 0x00D6
Pad Configuration Register 76	PCR76	16-bit	Base + 0x00D8
Pad Configuration Register 77	PCR77	16-bit	Base + 0x00DA
Pad Configuration Register 78	PCR78	16-bit	Base + 0x00DC
Pad Configuration Register 79	PCR79	16-bit	Base + 0x00DE
Pad Configuration Register 80	PCR80	16-bit	Base + 0x00E0
Pad Configuration Register 81	PCR81	16-bit	Base + 0x00E2
Pad Configuration Register 82	PCR82	16-bit	Base + 0x00E4
Pad Configuration Register 83	PCR83	16-bit	Base + 0x00E6
Pad Configuration Register 84	PCR84	16-bit	Base + 0x00E8
Pad Configuration Register 85	PCR85	16-bit	Base + 0x00EA
Pad Configuration Register 86	PCR86	16-bit	Base + 0x00EC
Pad Configuration Register 87	PCR87	16-bit	Base + 0x00EE
Pad Configuration Register 88	PCR88	16-bit	Base + 0x00F0
Pad Configuration Register 89	PCR89	16-bit	Base + 0x00F2
Pad Configuration Register 90	PCR90	16-bit	Base + 0x00F4
Pad Configuration Register 91	PCR91	16-bit	Base + 0x00F6
Pad Configuration Register 92	PCR92	16-bit	Base + 0x00F8
Pad Configuration Register 93	PCR93	16-bit	Base + 0x00FA
Pad Configuration Register 94	PCR94	16-bit	Base + 0x00FC
Pad Configuration Register 95	PCR95	16-bit	Base + 0x00FE
Pad Configuration Register 96	PCR96	16-bit	Base + 0x0100
Pad Configuration Register 97	PCR97	16-bit	Base + 0x0102
Pad Configuration Register 98	PCR98	16-bit	Base + 0x0104
Pad Configuration Register 99	PCR99	16-bit	Base + 0x0106
Pad Configuration Register 100	PCR100	16-bit	Base + 0x0108
Pad Configuration Register 101	PCR101	16-bit	Base + 0x010A
Pad Configuration Register 102	PCR102	16-bit	Base + 0x010C
Pad Configuration Register 103	PCR103	16-bit	Base + 0x010E
Pad Configuration Register 104	PCR104	16-bit	Base + 0x0110

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Pad Configuration Register 105	PCR105	16-bit	Base + 0x0112
Pad Configuration Register 106	PCR106	16-bit	Base + 0x0114
Pad Configuration Register 107	PCR107	16-bit	Base + 0x0116
Pad Configuration Register 108	PCR108	16-bit	Base + 0x0118
Pad Configuration Register 109	PCR109	16-bit	Base + 0x011A
Pad Configuration Register 110	PCR110	16-bit	Base + 0x011C
Pad Configuration Register 111	PCR111	16-bit	Base + 0x011E
Pad Configuration Register 112	PCR112	16-bit	Base + 0x0120
Pad Configuration Register 113	PCR113	16-bit	Base + 0x0122
Pad Configuration Register 114	PCR114	16-bit	Base + 0x0124
Pad Configuration Register 115	PCR115	16-bit	Base + 0x0126
Pad Configuration Register 116	PCR116	16-bit	Base + 0x0128
Pad Configuration Register 117	PCR117	16-bit	Base + 0x012A
Pad Configuration Register 118	PCR118	16-bit	Base + 0x012C
Pad Configuration Register 119	PCR119	16-bit	Base + 0x012E
Pad Configuration Register 120	PCR120	16-bit	Base + 0x0130
Pad Configuration Register 121	PCR121	16-bit	Base + 0x0132
Pad Configuration Register 122	PCR122	16-bit	Base + 0x0134
Reserved	—	—	Base + (0x0136 – 0x04FF)
Pad Selection for Multiplexed Inputs	PSMI0_3	32-bit	Base + 0x0500
Pad Selection for Multiplexed Inputs	PSMI4_7	32-bit	Base + 0x0504
Pad Selection for Multiplexed Inputs	PSMI8_11	32-bit	Base + 0x0508
Pad Selection for Multiplexed Inputs	PSMI12_15	32-bit	Base + 0x050C
Pad Selection for Multiplexed Inputs	PSMI16_19	32-bit	Base + 0x0510
Pad Selection for Multiplexed Inputs	PSMI20_23	32-bit	Base + 0x0514
Pad Selection for Multiplexed Inputs	PSMI24_27	32-bit	Base + 0x0518
Pad Selection for Multiplexed Inputs	PSMI28_31	32-bit	Base + 0x051C
Reserved	—	—	Base + (0x0520 – 0x05FF)
GPIO Pad Data Output Register	GPDO0_3	32-bit	Base + 0x0600
GPIO Pad Data Output Register	GPDO4_7	32-bit	Base + 0x0604
GPIO Pad Data Output Register	GPDO8_11	32-bit	Base + 0x0608
GPIO Pad Data Output Register	GPDO12_15	32-bit	Base + 0x060C
GPIO Pad Data Output Register	GPDO16_19	32-bit	Base + 0x0610

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
GPIO Pad Data Output Register	GPDO20_23	32-bit	Base + 0x0614
GPIO Pad Data Output Register	GPDO24_27	32-bit	Base + 0x0618
GPIO Pad Data Output Register	GPDO28_31	32-bit	Base + 0x061C
GPIO Pad Data Output Register	GPDO32_35	32-bit	Base + 0x0620
GPIO Pad Data Output Register	GPDO36_39	32-bit	Base + 0x0624
GPIO Pad Data Output Register	GPDO40_43	32-bit	Base + 0x0628
GPIO Pad Data Output Register	GPDO44_47	32-bit	Base + 0x062C
GPIO Pad Data Output Register	GPDO48_51	32-bit	Base + 0x0630
GPIO Pad Data Output Register	GPDO52_55	32-bit	Base + 0x0634
GPIO Pad Data Output Register	GPDO56_59	32-bit	Base + 0x0638
GPIO Pad Data Output Register	GPDO60_63	32-bit	Base + 0x063C
GPIO Pad Data Output Register	GPDO64_67	32-bit	Base + 0x0640
GPIO Pad Data Output Register	GPDO68_71	32-bit	Base + 0x0644
GPIO Pad Data Output Register	GPDO72_75	32-bit	Base + 0x0648
GPIO Pad Data Output Register	GPDO76_79	32-bit	Base + 0x064C
GPIO Pad Data Output Register	GPDO80_83	32-bit	Base + 0x0650
GPIO Pad Data Output Register	GPDO84_87	32-bit	Base + 0x0654
GPIO Pad Data Output Register	GPDO88_91	32-bit	Base + 0x0658
GPIO Pad Data Output Register	GPDO92_95	32-bit	Base + 0x065C
GPIO Pad Data Output Register	GPDO96_99	32-bit	Base + 0x0660
GPIO Pad Data Output Register	GPDO100_103	32-bit	Base + 0x0664
GPIO Pad Data Output Register	GPDO104_107	32-bit	Base + 0x0668
GPIO Pad Data Output Register	GPDO108_111	32-bit	Base + 0x066C
GPIO Pad Data Output Register	GPDO112_115	32-bit	Base + 0x0670
GPIO Pad Data Output Register	GPDO116_119	32-bit	Base + 0x0674
GPIO Pad Data Output Register	GPDO120_123	32-bit	Base + 0x0678
Reserved	—	—	Base + (0x067C – 0x07FF)
GPIO Pad Data Input Register	GPDI0_3	32-bit	Base + 0x0800
GPIO Pad Data Input Register	GPDI4_7	32-bit	Base + 0x0804
GPIO Pad Data Input Register	GPDI8_11	32-bit	Base + 0x0808
GPIO Pad Data Input Register	GPDI12_15	32-bit	Base + 0x080C
GPIO Pad Data Input Register	GPDI16_19	32-bit	Base + 0x0810
GPIO Pad Data Input Register	GPDI20_23	32-bit	Base + 0x0814

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
GPIO Pad Data Input Register	GPDI24_27	32-bit	Base + 0x0818
GPIO Pad Data Input Register	GPDI28_31	32-bit	Base + 0x081C
GPIO Pad Data Input Register	GPDI32_35	32-bit	Base + 0x0820
GPIO Pad Data Input Register	GPDI36_39	32-bit	Base + 0x0824
GPIO Pad Data Input Register	GPDI40_43	32-bit	Base + 0x0828
GPIO Pad Data Input Register	GPDI44_47	32-bit	Base + 0x082C
GPIO Pad Data Input Register	GPDI48_51	32-bit	Base + 0x0830
GPIO Pad Data Input Register	GPDI52_55	32-bit	Base + 0x0834
GPIO Pad Data Input Register	GPDI56_59	32-bit	Base + 0x0838
GPIO Pad Data Input Register	GPDI60_63	32-bit	Base + 0x083C
GPIO Pad Data Input Register	GPDI64_67	32-bit	Base + 0x0840
GPIO Pad Data Input Register	GPDI68_71	32-bit	Base + 0x0844
GPIO Pad Data Input Register	GPDI72_75	32-bit	Base + 0x0848
GPIO Pad Data Input Register	GPDI76_79	32-bit	Base + 0x084C
GPIO Pad Data Input Register	GPDI80_83	32-bit	Base + 0x0850
GPIO Pad Data Input Register	GPDI84_87	32-bit	Base + 0x0854
GPIO Pad Data Input Register	GPDI88_91	32-bit	Base + 0x0858
GPIO Pad Data Input Register	GPDI92_95	32-bit	Base + 0x085C
GPIO Pad Data Input Register	GPDI96_99	32-bit	Base + 0x0860
GPIO Pad Data Input Register	GPDI100_103	32-bit	Base + 0x0864
GPIO Pad Data Input Register	GPDI104_107	32-bit	Base + 0x0868
GPIO Pad Data Input Register	GPDI108_111	32-bit	Base + 0x086C
GPIO Pad Data Input Register	GPDI112_115	32-bit	Base + 0x0870
GPIO Pad Data Input Register	GPDI116_119	32-bit	Base + 0x0874
GPIO Pad Data Input Register	GPDI120_123	32-bit	Base + 0x0878
Reserved	—	—	Base + (0x087C – 0x0BFF)
Parallel GPIO Pad Data Out Register	PGPDO0	32-bit	Base + 0x0C00
Parallel GPIO Pad Data Out Register	PGPDO1	32-bit	Base + 0x0C04
Parallel GPIO Pad Data Out Register	PGPDO2	32-bit	Base + 0x0C08
Parallel GPIO Pad Data Out Register	PGPDO3	32-bit	Base + 0x0C0C
Reserved	—	—	(Base + 0x0C10) – (Base + 0x0C3F)
Parallel GPIO Pad Data In Register	PGPDI0	32-bit	Base + 0x0C40
Parallel GPIO Pad Data In Register	PGPDI1	32-bit	Base + 0x0C44

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Parallel GPIO Pad Data In Register	PGPDI2	32-bit	Base + 0x0C48
Parallel GPIO Pad Data In Register	PGPDI3	32-bit	Base + 0x0C4C
Reserved	—	—	(Base + 0x0C50) – (Base + 0x0C7F)
Masked Parallel GPIO Pad Data Out Register	MPGPDO0	32-bit	Base + 0x0C80
Masked Parallel GPIO Pad Data Out Register	MPGPDO1	32-bit	Base + 0x0C84
Masked Parallel GPIO Pad Data Out Register	MPGPDO2	32-bit	Base + 0x0C88
Masked Parallel GPIO Pad Data Out Register	MPGPDO3	32-bit	Base + 0x0C8C
Masked Parallel GPIO Pad Data Out Register	MPGPDO4	32-bit	Base + 0x0C90
Masked Parallel GPIO Pad Data Out Register	MPGPDO5	32-bit	Base + 0x0C94
Masked Parallel GPIO Pad Data Out Register	MPGPDO6	32-bit	Base + 0x0C98
Masked Parallel GPIO Pad Data Out Register	MPGPDO7	32-bit	Base + 0x0C9C
Reserved	—	—	Base + (0x0CA0 – 0x0FFF)
Interrupt Filter Maximum Counter Register	IFMC0	32-bit	Base + 0x1000
Interrupt Filter Maximum Counter Register	IFMC1	32-bit	Base + 0x1004
Interrupt Filter Maximum Counter Register	IFMC2	32-bit	Base + 0x1008
Interrupt Filter Maximum Counter Register	IFMC3	32-bit	Base + 0x100C
Interrupt Filter Maximum Counter Register	IFMC4	32-bit	Base + 0x1010
Interrupt Filter Maximum Counter Register	FMC5	32-bit	Base + 0x1014
Interrupt Filter Maximum Counter Register	IFMC6	32-bit	Base + 0x1018
Interrupt Filter Maximum Counter Register	IFMC7	32-bit	Base + 0x101C
Interrupt Filter Maximum Counter Register	IFMC8	32-bit	Base + 0x1020
Interrupt Filter Maximum Counter Register	IFMC9	32-bit	Base + 0x1024
Interrupt Filter Maximum Counter Register	IFMC10	32-bit	Base + 0x1028
Interrupt Filter Maximum Counter Register	IFMC11	32-bit	Base + 0x102C
Interrupt Filter Maximum Counter Register	IFMC12	32-bit	Base + 0x1030
Interrupt Filter Maximum Counter Register	IFMC13	32-bit	Base + 0x1034
Interrupt Filter Maximum Counter Register	IFMC14	32-bit	Base + 0x1038
Interrupt Filter Maximum Counter Register	IFMC15	32-bit	Base + 0x103C
Reserved	—	—	(Base + 0x1044 – 0x107C)
Inerrupt Filter Clock Prescaler Register	IFCP	32-bit	Base + 0x1080
Reserved	—	—	Base + (0x1084 – 0x3FFF)

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
WakeUp Unit			**0xC3F9_4000**
NMI Status Flag Register	WKPU_NSR	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x0007)
NMI Configuration Register	WKPU_NCR	32-bit	Base + 0x0008
Reserved	—	—	(Base + 0x000C) – (Base + 0x0013)
Wakeup/Interrupt Status Flag Register	WKPU_WISR	32-bit	Base + 0x0014
Interrupt Request Enable Register	WKPU_IRER	32-bit	Base + 0x0018
Wakeup Request Enable Register	WKPU_WRER	32-bit	Base + 0x001C
Reserved	—	—	(Base + 0x0020) – (Base + 0x0027)
Wakeup/Interrupt Rising-Edge Event Enable Register	WKPU_WIREER	32-bit	Base + 0x0028
Wakeup/Interrupt Falling-Edge Event Enable Register	WKPU_WIFEER	32-bit	Base + 0x002C
Wakeup/Interrupt Filter Enable Register	WKPU_WIFER	32-bit	Base + 0x0030
Wakeup/Interrupt Pullup Enable Register	WKPU_WIPUER	32-bit	Base + 0x0034
Reserved	—	—	(Base + 0x0038) – (Base + 0xFFFF)
eMIOS_0			**0xC3FA_0000**
EMIOS Module Configuration Register	EMIOS0_MCR	32-bit	Base + 0x0000
EMIOS Global FLAG Register	EMIOS0_GFLAG	32-bit	Base + 0x0004
EMIOS Output Update Disable Register	EMIOS0_OUDIS	32-bit	Base + 0x0008
EMIOS Disable Channel Register	EMIOS0_UCDIS	32-bit	Base + 0x000C
Reserved	—	—	(Base + 0x0010) – (Base + 0x001F)
eMIOS_0 UC0 A Register	EMIOS0_UC0_A	32-bit	Base + 0x0020
eMIOS_0 UC0 B Register	EMIOS0_UC0_B	32-bit	Base + 0x0024
eMIOS_0 UC0 CNT	EMIOS0_UC0_CNT	32-bit	Base + 0x0028
eMIOS_0 UC0 Control Register	EMIOS0_UC0_SC	32-bit	Base + 0x002C
eMIOS_0 UC0 Status Register	EMIOS0_UC0_SS	32-bit	Base + 0x0030
Reserved	—	—	Base + 0x0034 – Base + 0x003F
eMIOS_0 UC1 A Register	EMIOS0_UC1_A	32-bit	Base + 0x0040
eMIOS_0 UC1 B Register	EMIOS0_UC1_B	32-bit	Base + 0x0044
Reserved	—	—	Base + 0x0048 -– Base + 0x004B
eMIOS_0 UC1 Control Register	EMIOS0_UC1_SC	32-bit	Base + 0x004C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_0 UC1 Status Register	EMIOS0_UC1_SS	32-bit	Base + 0x0050
Reserved	—	—	Base + 0x0054 – Base + 0x005F
eMIOS_0 UC2 A Register	EMIOS0_UC2_A	32-bit	Base + 0x0060
eMIOS_0 UC2 B Register	EMIOS0_UC2_B	32-bit	Base + 0x0064
Reserved	—	—	Base + 0x0068 – Base + 0x006B
eMIOS_0 UC2 Control Register	EMIOS0_UC2_SC	32-bit	Base + 0x006C
eMIOS_0 UC2 Status Register	EMIOS0_UC2_SS	32-bit	Base + 0x0070
Reserved	—	—	Base + 0x0074 – Base + 0x007F
eMIOS_0 UC3 A Register	EMIOS0_UC3_A	32-bit	Base + 0x0080
eMIOS_0 UC3 B Register	EMIOS0_UC3_B	32-bit	Base + 0x0084
Reserved	—	—	Base + 0x0088 – Base + 0x008B
eMIOS_0 UC3 Control Register	EMIOS0_UC3_SC	32-bit	Base + 0x008C
eMIOS_0 UC3 Status Register	EMIOS0_UC3_SS	32-bit	Base + 0x0090
Reserved	—	—	Base + 0x0094 – Base + 0x009F
eMIOS_0 UC4 A Register	EMIOS0_UC4_A	32-bit	Base + 0x00A0
eMIOS_0 UC4 B Register	EMIOS0_UC4_B	32-bit	Base + 0x00A4
Reserved	—	—	Base + 0x00A8 – Base + 0x00AB
eMIOS_0 UC4 Control Register	EMIOS0_UC4_SC	32-bit	Base + 0x00AC
eMIOS_0 UC4 Status Register	EMIOS0_UC4_SS	32-bit	Base + 0x00B0
Reserved	—	—	Base + 0x00B4 – Base + 0x00BF
eMIOS_0 UC5 A Register	EMIOS0_UC5_A	32-bit	Base + 0x00C0
eMIOS_0 UC5 B Register	EMIOS0_UC5_B	32-bit	Base + 0x00C4
Reserved	—	—	Base + 0x00C8 – Base + 0x00CB
eMIOS_0 UC5 Control Register	EMIOS0_UC5_SC	32-bit	Base + 0x00CC
eMIOS_0 UC5 Status Register	EMIOS0_UC5_SS	32-bit	Base + 0x00D0
Reserved	—	—	Base + 0x00D4 – Base + 0x00DF
eMIOS_0 UC6 A Register	EMIOS0_UC6_A	32-bit	Base + 0x00E0
eMIOS_0 UC6 B Register	EMIOS0_UC6_B	32-bit	Base + 0x00E4

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + 0x00E8 – Base + 0x00EB
eMIOS_0 UC6 Control Register	EMIOS0_UC6_SC	32-bit	Base + 0x00EC
eMIOS_0 UC6 Status Register	EMIOS0_UC6_SS	32-bit	Base + 0x00F0
Reserved	—	—	Base + 0x00F4 – Base + 0x00FF
eMIOS_0 UC7 A Register	EMIOS0_UC7_A	32-bit	Base + 0x0100
eMIOS_0 UC7 B Register	EMIOS0_UC7_B	32-bit	Base + 0x0104
Reserved	—	—	Base + 0x0108 – Base + 0x010B
eMIOS_0 UC7 Control Register	EMIOS0_UC7_SC	32-bit	Base + 0x010C
eMIOS_0 UC7 Status Register	EMIOS0_UC7_SS	32-bit	Base + 0x0110
Reserved	—	—	Base + 0x0114 – Base + 0x011F
eMIOS_0 UC8 A Register	EMIOS0_UC8_A	32-bit	Base + 0x0120
eMIOS_0 UC8 B Register	EMIOS0_UC8_B	32-bit	Base + 0x0124
eMIOS_0 UC8 CNT	EMIOS0_UC8_CNT	32-bit	Base + 0x0128
eMIOS_0 UC8 Control Register	EMIOS0_UC8_SC	32-bit	Base + 0x012C
eMIOS_0 UC8 Status Register	EMIOS0_UC8_SS	32-bit	Base + 0x0130
Reserved	—	—	Base + 0x0134 – Base + 0x013F
eMIOS_0 UC9 A Register	EMIOS0_UC9_A	32-bit	Base + 0x0140
eMIOS_0 UC9 B Register	EMIOS0_UC9_B	32-bit	Base + 0x0144
Reserved	—	—	Base + 0x0148 – Base + 0x014B
eMIOS_0 UC9 Control Register	EMIOS0_UC9_SC	32-bit	Base + 0x014C
eMIOS_0 UC9 Status Register	EMIOS0_UC9_SS	32-bit	Base + 0x0150
Reserved	—	—	Base + 0x0154 – Base + 0x015F
eMIOS_0 UC10 A Register	EMIOS0_UC10_A	32-bit	Base + 0x0160
eMIOS_0 UC10 B Register	EMIOS0_UC10_B	32-bit	Base + 0x0164
Reserved	—	—	Base + 0x0168 – Base + 0x016B
eMIOS_0 UC10 Control Register	EMIOS0_UC10_SC	32-bit	Base + 0x016C
eMIOS_0 UC10 Status Register	EMIOS0_UC10_SS	32-bit	Base + 0x0170
Reserved	—	—	Base + 0x0174 – Base + 0x017F

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_0 UC11 A Register	EMIOS0_UC11_A	32-bit	Base + 0x0180
eMIOS_0 UC11 B Register	EMIOS0_UC11_B	32-bit	Base + 0x0184
Reserved	—	—	Base + 0x0188 – Base + 0x018B
eMIOS_0 UC11 Control Register	EMIOS0_UC11_SC	32-bit	Base + 0x018C
eMIOS_0 UC11 Status Register	EMIOS0_UC11_SS	32-bit	Base + 0x0190
Reserved	—	—	Base + 0x0194 – Base + 0x019F
eMIOS_0 UC12 A Register	EMIOS0_UC12_A	32-bit	Base + 0x01A0
eMIOS_0 UC12 B Register	EMIOS0_UC12_B	32-bit	Base + 0x01A4
Reserved	—	—	Base + 0x01A8 – Base + 0x01AB
eMIOS_0 UC12 Control Register	EMIOS0_UC12_SC	32-bit	Base + 0x01AC
eMIOS_0 UC12 Status Register	EMIOS0_UC12_SS	32-bit	Base + 0x01B0
Reserved	—	—	Base + 0x01B4 – Base + 0x01BF
eMIOS_0 UC13 A Register	EMIOS0_UC13_A	32-bit	Base + 0x01C0
eMIOS_0 UC13 B Register	EMIOS0_UC13_B	32-bit	Base + 0x01C4
Reserved	—	—	Base + 0x01C8 – Base + 0x01CB
eMIOS_0 UC13 Control Register	EMIOS0_UC13_SC	32-bit	Base + 0x01CC
eMIOS_0 UC13 Status Register	EMIOS0_UC13_SS	32-bit	Base + 0x01D0
Reserved	—	—	Base + 0x01D4 – Base + 0x01DF
eMIOS_0 UC14 A Register	EMIOS0_UC14_A	32-bit	Base + 0x01E0
eMIOS_0 UC14 B Register	EMIOS0_UC14_B	32-bit	Base + 0x01E4
Reserved	—	—	Base + 0x01E8 – Base + 0x01EB
eMIOS_0 UC14 Control Register	EMIOS0_UC14_SC	32-bit	Base + 0x01EC
eMIOS_0 UC14 Status Register	EMIOS0_UC14_SS	32-bit	Base + 0x01F0
Reserved	—	—	Base + 0x01F4 – Base + 0x01FF
eMIOS_0 UC15 A Register	EMIOS0_UC15_A	32-bit	Base + 0x0200
eMIOS_0 UC15 B Register	EMIOS0_UC15_B	32-bit	Base + 0x0204
Reserved	—	—	Base + 0x0208 – Base + 0x020B
eMIOS_0 UC15 Control Register	EMIOS0_UC15_SC	32-bit	Base + 0x020C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_0 UC15 Status Register	EMIOS0_UC15_SS	32-bit	Base + 0x0210
Reserved	—	—	Base + 0x0214 – Base + 0x021F
eMIOS_0 UC16 A Register	EMIOS0_UC16_A	32-bit	Base + 0x0220
eMIOS_0 UC16 B Register	EMIOS0_UC16_B	32-bit	Base + 0x0224
eMIOS_0 UC16 CNT	EMIOS0_UC16_CNT	32-bit	Base + 0x0228
eMIOS_0 UC16 Control Register	EMIOS0_UC16_SC	32-bit	Base + 0x022C
eMIOS_0 UC16 Status Register	EMIOS0_UC16_SS	32-bit	Base + 0x0230
Reserved	—	—	Base + 0x0234 – Base + 0x023F
eMIOS_0 UC17 A Register	EMIOS0_UC17_A	32-bit	Base + 0x0240
eMIOS_0 UC17 B Register	EMIOS0_UC17_B	32-bit	Base + 0x0244
Reserved	—	—	Base + 0x0248 – Base + 0x024B
eMIOS_0 UC17 Control Register	EMIOS0_UC17_SC	32-bit	Base + 0x024C
eMIOS_0 UC17 Status Register	EMIOS0_UC17_SS	32-bit	Base + 0x0250
Reserved	—	—	Base + 0x0254 – Base + 0x025F
eMIOS_0 UC18 A Register	EMIOS0_UC18_A	32-bit	Base + 0x0260
eMIOS_0 UC18 B Register	EMIOS0_UC18_B	32-bit	Base + 0x0264
Reserved	—	—	Base + 0x0268 – Base + 0x026B
eMIOS_0 UC18 Control Register	EMIOS0_UC18_SC	32-bit	Base + 0x026C
eMIOS_0 UC18 Status Register	EMIOS0_UC18_SS	32-bit	Base + 0x0270
Reserved	—	—	Base + 0x0274 – Base + 0x027F
eMIOS_0 UC19 A Register	EMIOS0_UC19_A	32-bit	Base + 0x0280
eMIOS_0 UC19 B Register	EMIOS0_UC19_B	32-bit	Base + 0x0284
Reserved	—	—	Base + 0x0288 – Base + 0x028B
eMIOS_0 UC19 Control Register	EMIOS0_UC19_SC	32-bit	Base + 0x028C
eMIOS_0 UC19 Status Register	EMIOS0_UC19_SS	32-bit	Base + 0x0290
Reserved	—	—	Base + 0x0294 – Base + 0x029F
eMIOS_0 UC20 A Register	EMIOS0_UC20_A	32-bit	Base + 0x02A0
eMIOS_0 UC20 B Register	EMIOS0_UC20_B	32-bit	Base + 0x02A4

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + 0x02A8 – Base + 0x02AB
eMIOS_0 UC20 Control Register	EMIOS0_UC20_SC	32-bit	Base + 0x02AC
eMIOS_0 UC20 Status Register	EMIOS0_UC20_SS	32-bit	Base + 0x02B0
Reserved	—	—	Base + 0x02B4 – Base + 0x02BF
eMIOS_0 UC21 A Register	EMIOS0_UC21_A	32-bit	Base + 0x02C0
eMIOS_0 UC21 B Register	EMIOS0_UC21_B	32-bit	Base + 0x02C4
Reserved	—	—	Base + 0x02C8 – Base + 0x02CB
eMIOS_0 UC21 Control Register	EMIOS0_UC21_SC	32-bit	Base + 0x02CC
eMIOS_0 UC21 Status Register	EMIOS0_UC21_SS	32-bit	Base + 0x02D0
Reserved	—	—	Base + 0x02D4 – Base + 0x02DF
eMIOS_0 UC22 A Register	EMIOS0_UC22_A	32-bit	Base + 0x02E0
eMIOS_0 UC22 B Register	EMIOS0_UC22_B	32-bit	Base + 0x02E4
Reserved	—	—	Base + 0x02E8 – Base + 0x02EB
eMIOS_0 UC22 Control Register	EMIOS0_UC22_SC	32-bit	Base + 0x02EC
eMIOS_0 UC22 Status Register	EMIOS0_UC22_SS	32-bit	Base + 0x02F0
Reserved	—	—	Base + 0x02F4 – Base + 0x02FF
eMIOS_0 UC23 A Register	EMIOS0_UC23_A	32-bit	Base + 0x0300
eMIOS_0 UC23 B Register	EMIOS0_UC23_B	32-bit	Base + 0x0304
eMIOS_0 UC23 CNT	EMIOS0_UC23_CNT	32-bit	Base + 0x0308
eMIOS_0 UC23 Control Register	EMIOS0_UC23_SC	32-bit	Base + 0x030C
eMIOS_0 UC23 Status Register	EMIOS0_UC23_SS	32-bit	Base + 0x0310
Reserved	—	—	Base + 0x0314 – Base + 0x031F
eMIOS_1			**0xC3FA_4000**
EMIOS Module Configuration Register	eMIOS1_MCR	32-bit	Base + 0x0000
EMIOS Global FLAG Register	eMIOS1_GFLAG	32-bit	Base + 0x0004
EMIOS Output Update Disable Register	eMIOS1_OUDIS	32-bit	Base + 0x0008
EMIOS Disable Channel Register	eMIOS1_UCDIS	32-bit	Base + 0x000C
Reserved	-	-	(Base + 0x001C) – (Base + 0x001F)
eMIOS_1 UC0 A Register	eMIOS1_UC0_A	32-bit	Base + 0x0020

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_1 UC0 B Register	eMIOS1_UC0_B	32-bit	Base + 0x0024
eMIOS_1 UC0 CNT	eMIOS1_UC0_CNT	32-bit	Base + 0x0028
eMIOS_1 UC0 Control Register	eMIOS1_UC0_SC	32-bit	Base + 0x002C
eMIOS_1 UC0 Status Register	eMIOS1_UC0_SS	32-bit	Base + 0x0030
Reserved	—	—	Base + 0x0034 – Base + 0x003F
eMIOS_1 UC1 A Register	eMIOS1_UC1_A	32-bit	Base + 0x0040
eMIOS_1 UC1 B Register	eMIOS1_UC1_B	32-bit	Base + 0x0044
Reserved	—	—	Base + 0x0048 – Base + 0x004B
eMIOS_1 UC1 Control Register	eMIOS1_UC1_SC	32-bit	Base + 0x004C
eMIOS_1 UC1 Status Register	eMIOS1_UC1_SS	32-bit	Base + 0x0050
Reserved	—	—	Base + 0x0054 – Base + 0x005F
eMIOS_1 UC2 A Register	eMIOS1_UC2_A	32-bit	Base + 0x0060
eMIOS_1 UC2 B Register	eMIOS1_UC2_B	32-bit	Base + 0x0064
Reserved	—	—	Base + 0x0068 – Base + 0x006B
eMIOS_1 UC2 Control Register	eMIOS1_UC2_SC	32-bit	Base + 0x006C
eMIOS_1 UC2 Status Register	eMIOS1_UC2_SS	32-bit	Base + 0x0070
Reserved	—	—	Base + 0x0074 – Base + 0x007F
eMIOS_1 UC3 A Register	eMIOS1_UC3_A	32-bit	Base + 0x0080
eMIOS_1 UC3 B Register	eMIOS1_UC3_B	32-bit	Base + 0x0084
Reserved	—	—	Base + 0x0088 – Base + 0x008B
eMIOS_1 UC3 Control Register	eMIOS1_UC3_SC	32-bit	Base + 0x008C
eMIOS_1 UC3 Status Register	eMIOS1_UC3_SS	32-bit	Base + 0x0090
Reserved	—	—	Base + 0x0094 – Base + 0x009F
eMIOS_1 UC4 A Register	eMIOS1_UC4_A	32-bit	Base + 0x00A0
eMIOS_1 UC4 B Register	eMIOS1_UC4_B	32-bit	Base + 0x00A4
Reserved	—	—	Base + 0x00A8 – Base + 0x00AB
eMIOS_1 UC4 Control Register	eMIOS1_UC4_SC	32-bit	Base + 0x00AC
eMIOS_1 UC4 Status Register	eMIOS1_UC4_SS	32-bit	Base + 0x00B0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + 0x00B4 – Base + 0x00BF
eMIOS_1 UC5 A Register	eMIOS1_UC5_A	32-bit	Base + 0x00C0
eMIOS_1 UC5 B Register	eMIOS1_UC5_B	32-bit	Base + 0x00C4
Reserved	—	—	Base + 0x00C8 – Base + 0x00CB
eMIOS_1 UC5 Control Register	eMIOS1_UC5_SC	32-bit	Base + 0x00CC
eMIOS_1 UC5 Status Register	eMIOS1_UC5_SS	32-bit	Base + 0x00D0
Reserved	—	—	Base + 0x00D4 – Base + 0x00DF
eMIOS_1 UC6 A Register	eMIOS1_UC6_A	32-bit	Base + 0x00E0
eMIOS_1 UC6 B Register	eMIOS1_UC6_B	32-bit	Base + 0x00E4
Reserved	—	—	Base + 0x00E8 – Base + 0x00EB
eMIOS_1 UC6 Control Register	eMIOS1_UC6_SC	32-bit	Base + 0x00EC
eMIOS_1 UC6 Status Register	eMIOS1_UC6_SS	32-bit	Base + 0x00F0
Reserved	—	—	Base + 0x00F4 – Base + 0x00FF
eMIOS_1 UC7 A Register	eMIOS1_UC7_A	32-bit	Base + 0x0100
eMIOS_1 UC7 B Register	eMIOS1_UC7_B	32-bit	Base + 0x0104
Reserved	—	—	Base + 0x0108 – Base + 0x010B
eMIOS_1 UC7 Control Register	eMIOS1_UC7_SC	32-bit	Base + 0x010C
eMIOS_1 UC7 Status Register	eMIOS1_UC7_SS	32-bit	Base + 0x0110
Reserved	—	—	Base + 0x0114 – Base + 0x011F
eMIOS_1 UC8 A Register	eMIOS1_UC8_A	32-bit	Base + 0x0120
eMIOS_1 UC8 B Register	eMIOS1_UC8_B	32-bit	Base + 0x0124
eMIOS_1 UC8 CNT	eMIOS1_UC8_CNT	32-bit	Base + 0x0128
eMIOS_1 UC8 Control Register	eMIOS1_UC8_SC	32-bit	Base + 0x012C
eMIOS_1 UC8 Status Register	eMIOS1_UC8_SS	32-bit	Base + 0x0130
Reserved	—	—	Base + 0x0134 – Base + 0x013F
eMIOS_1 UC9 A Register	eMIOS1_UC9_A	32-bit	Base + 0x0140
eMIOS_1 UC9 B Register	eMIOS1_UC9_B	32-bit	Base + 0x0144
Reserved	—	—	Base + 0x0148 – Base + 0x014B

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_1 UC9 Control Register	eMIOS1_UC9_SC	32-bit	Base + 0x014C
eMIOS_1 UC9 Status Register	eMIOS1_UC9_SS	32-bit	Base + 0x0150
Reserved	—	—	Base + 0x0154 – Base + 0x015F
eMIOS_1 UC10 A Register	eMIOS1_UC10_A	32-bit	Base + 0x0160
eMIOS_1 UC10 B Register	eMIOS1_UC10_B	32-bit	Base + 0x0164
Reserved	—	—	Base + 0x0168 – Base + 0x016B
eMIOS_1 UC10 Control Register	eMIOS1_UC10_SC	32-bit	Base + 0x016C
eMIOS_1 UC10 Status Register	eMIOS1_UC10_SS	32-bit	Base + 0x0170
Reserved	—	—	Base + 0x0174 – Base + 0x017F
eMIOS_1 UC11 A Register	eMIOS1_UC11_A	32-bit	Base + 0x0180
eMIOS_1 UC11 B Register	eMIOS1_UC11_B	32-bit	Base + 0x0184
Reserved	—	—	Base + 0x0188 – Base + 0x018B
eMIOS_1 UC11 Control Register	eMIOS1_UC11_SC	32-bit	Base + 0x018C
eMIOS_1 UC11 Status Register	eMIOS1_UC11_SS	32-bit	Base + 0x0190
Reserved	—	—	Base + 0x0194 – Base + 0x019F
eMIOS_1 UC12 A Register	eMIOS1_UC12_A	32-bit	Base + 0x01A0
eMIOS_1 UC12 B Register	eMIOS1_UC12_B	32-bit	Base + 0x01A4
Reserved	—	—	Base + 0x01A8 – Base + 0x01AB
eMIOS_1 UC12 Control Register	eMIOS1_UC12_SC	32-bit	Base + 0x01AC
eMIOS_1 UC12 Status Register	eMIOS1_UC12_SS	32-bit	Base + 0x01B0
Reserved	—	—	Base + 0x01B4 – Base + 0x01BF
eMIOS_1 UC13 A Register	eMIOS1_UC13_A	32-bit	Base + 0x01C0
eMIOS_1 UC13 B Register	eMIOS1_UC13_B	32-bit	Base + 0x01C4
Reserved	—	—	Base + 0x01C8 – Base + 0x01CB
eMIOS_1 UC13 Control Register	eMIOS1_UC13_SC	32-bit	Base + 0x01CC
eMIOS_1 UC13 Status Register	eMIOS1_UC13_SS	32-bit	Base + 0x01D0
Reserved	—	—	Base + 0x01D4 – Base + 0x01DF
eMIOS_1 UC14 A Register	eMIOS1_UC14_A	32-bit	Base + 0x01E0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_1 UC14 B Register	eMIOS1_UC14_B	32-bit	Base + 0x01E4
Reserved	—	—	Base + 0x01E8 – Base + 0x01EB
eMIOS_1 UC14 Control Register	eMIOS1_UC14_SC	32-bit	Base + 0x01EC
eMIOS_1 UC14 Status Register	eMIOS1_UC14_SS	32-bit	Base + 0x01F0
Reserved	—	—	Base + 0x01F4 – Base + 0x01FF
eMIOS_1 UC15 A Register	eMIOS1_UC15_A	32-bit	Base + 0x0200
eMIOS_1 UC15 B Register	eMIOS1_UC15_B	32-bit	Base + 0x0204
Reserved	—	—	Base + 0x0208 – Base + 0x020B
eMIOS_1 UC15 Control Register	eMIOS1_UC15_SC	32-bit	Base + 0x020C
eMIOS_1 UC15 Status Register	eMIOS1_UC15_SS	32-bit	Base + 0x0210
Reserved	—	—	Base + 0x0214 – Base + 0x021F
eMIOS_1 UC16 A Register	eMIOS1_UC16_A	32-bit	Base + 0x0220
eMIOS_1 UC16 B Register	eMIOS1_UC16_B	32-bit	Base + 0x0224
eMIOS_1 UC16 CNT	eMIOS1_UC16_CNT	32-bit	Base + 0x0228
eMIOS_1 UC16 Control Register	eMIOS1_UC16_SC	32-bit	Base + 0x022C
eMIOS_1 UC16 Status Register	eMIOS1_UC16_SS	32-bit	Base + 0x0230
Reserved	—	—	Base + 0x0234 – Base + 0x023F
eMIOS_1 UC17 A Register	eMIOS1_UC17_A	32-bit	Base + 0x0240
eMIOS_1 UC17 B Register	eMIOS1_UC17_B	32-bit	Base + 0x0244
Reserved	—	—	Base + 0x0248 – Base + 0x024B
eMIOS_1 UC17 Control Register	eMIOS1_UC17_SC	32-bit	Base + 0x024C
eMIOS_1 UC17 Status Register	eMIOS1_UC17_SS	32-bit	Base + 0x0250
Reserved	—	—	Base + 0x0254 – Base + 0x025F
eMIOS_1 UC18 A Register	eMIOS1_UC18_A	32-bit	Base + 0x0260
eMIOS_1 UC18 B Register	eMIOS1_UC18_B	32-bit	Base + 0x0264
Reserved	—	—	Base + 0x0268 – Base + 0x026B
eMIOS_1 UC18 Control Register	eMIOS1_UC18_SC	32-bit	Base + 0x026C
eMIOS_1 UC18 Status Register	eMIOS1_UC18_SS	32-bit	Base + 0x0270

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + 0x0274 – Base + 0x027F
eMIOS_1 UC19 A Register	eMIOS1_UC19_A	32-bit	Base + 0x0280
eMIOS_1 UC19 B Register	eMIOS1_UC19_B	32-bit	Base + 0x0284
Reserved	—	—	Base + 0x0288 – Base + 0x028B
eMIOS_1 UC19 Control Register	eMIOS1_UC19_SC	32-bit	Base + 0x028C
eMIOS_1 UC19 Status Register	eMIOS1_UC19_SS	32-bit	Base + 0x0290
Reserved	—	—	Base + 0x0294 – Base + 0x029F
eMIOS_1 UC20 A Register	eMIOS1_UC20_A	32-bit	Base + 0x02A0
eMIOS_1 UC20 B Register	eMIOS1_UC20_B	32-bit	Base + 0x02A4
Reserved	—	—	Base + 0x02A8 – Base + 0x02AB
eMIOS_1 UC20 Control Register	eMIOS1_UC20_SC	32-bit	Base + 0x02AC
eMIOS_1 UC20 Status Register	eMIOS1_UC20_SS	32-bit	Base + 0x02B0
Reserved	—	—	Base + 0x02B4 – Base + 0x02BF
eMIOS_1 UC21 A Register	eMIOS1_UC21_A	32-bit	Base + 0x02C0
eMIOS_1 UC21 B Register	eMIOS1_UC21_B	32-bit	Base + 0x02C4
Reserved	—	—	Base + 0x02C8 – Base + 0x02CB
eMIOS_1 UC21 Control Register	eMIOS1_UC21_SC	32-bit	Base + 0x02CC
eMIOS_1 UC21 Status Register	eMIOS1_UC21_SS	32-bit	Base + 0x02D0
Reserved	—	—	Base + 0x02D4 – Base + 0x02DF
eMIOS_1 UC22 A Register	eMIOS1_UC22_A	32-bit	Base + 0x02E0
eMIOS_1 UC22 B Register	eMIOS1_UC22_B	32-bit	Base + 0x02E4
Reserved	—	—	Base + 0x02E8 – Base + 0x02EB
eMIOS_1 UC22 Control Register	eMIOS1_UC22_SC	32-bit	Base + 0x02EC
eMIOS_1 UC22 Status Register	eMIOS1_UC22_SS	32-bit	Base + 0x02F0
Reserved	—	—	Base + 0x02F4 – Base + 0x02FF
eMIOS_1 UC23 A Register	eMIOS1_UC23_A	32-bit	Base + 0x0300
eMIOS_1 UC23 B Register	eMIOS1_UC23_B	32-bit	Base + 0x0304
eMIOS_1 UC23 CNT	eMIOS1_UC23_CNT	32-bit	Base + 0x0308

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
eMIOS_1 UC23 Control Register	eMIOS1_UC23_SC	32-bit	Base + 0x030C
eMIOS_1 UC23 Status Register	eMIOS1_UC23_SS	32-bit	Base + 0x0310
System Status and Configuration Module (SSCM)			**0xC3FD_8000**
System Status Register	STATUS	16-bit	Base + 0x0000
System Memory Configuration Register	MEMCONFIG	16-bit	Base + 0x0002
Reserved	-	-	Base + (0x0004 – 0x0005)
Error Configuration	ERROR	16-bit	Base + 0x0006
Reserved	-	-	Base + (0x0008 – 0x000B)
Password Comparison Register High Word	PWCMPH	32-bit	Base + 0x000C
Password Comparison Register Low Word	PWCMPL	32-bit	Base + 0x0010
Reserved	-	-	Base + (0x0014 – 0x3FFF)
Mode Entry Module (MC_ME)			**0xC3FD_C000**
Global Status	ME_GS	32-bit	Base + 0x0000
Mode Control	ME_MCTL	32-bit	Base + 0x0004
Mode Enable	ME_ME	32-bit	Base + 0x0008
Interrupt Status	ME_IS	32-bit	Base + 0x000C
Interrupt Mask	ME_IM	32-bit	Base + 0x0010
Invalid Mode Transition status	ME_IMTS	32-bit	Base + 0x0014
Debug Mode Transition status	ME_DMTS	32-bit	Base + 0x0018
RESET Mode Configuration	ME_RESET_MC	32-bit	Base + 0x0020
TEST Mode Configuration	ME_TEST_MC	32-bit	Base + 0x0024
SAFE Mode Configuration	ME_SAFE_MC	32-bit	Base + 0x0028
DRUN Mode Configuration	ME_DRUN_MC	32-bit	Base + 0x002C
RUN0 Mode Configuration	ME_RUN0_MC	32-bit	Base + 0x0030
RUN1 Mode Configuration	ME_RUN1_MC	32-bit	Base + 0x0034
RUN2 Mode Configuration	ME_RUN2_MC	32-bit	Base + 0x0038
RUN3 Mode Configuration	ME_RUN3_MC	32-bit	Base + 0x003C
HALT Mode Configuration	ME_HALT_MC	32-bit	Base + 0x0040
Reserved	—	—	Base + 0x0044 – Base + 0x0047
STOP Mode Configuration	ME_STOP_MC	32-bit	Base + 0x0048
Reserved	—	—	Base + 0x004C – Base + 0x0053

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
STANDBY Mode Configuration	ME_STANDBY_MC	32-bit	Base + 0x0054
Reserved	—	—	Base + 0x0058 – Base + 0x005F
Peripheral Status Registers	ME_PS0	32-bit	Base + 0x0060
Peripheral Status Registers	ME_PS1	32-bit	Base + 0x0064
Peripheral Status Registers	ME_PS2	32-bit	Base + 0x0068
Peripheral Status Registers	ME_PS3	32-bit	Base + 0x006C
Reserved	-	-	(Base + 0x0070) – (Base + 0x007F)
RUN Peripheral Configuration Registers	ME_RUN_PC0	32-bit	Base + 0x0080
RUN Peripheral Configuration Registers	ME_RUN_PC1	32-bit	Base + 0x0084
RUN Peripheral Configuration Registers	ME_RUN_PC2	32-bit	Base + 0x0088
RUN Peripheral Configuration Registers	ME_RUN_PC3	32-bit	Base + 0x008C
RUN Peripheral Configuration Registers	ME_RUN_PC4	32-bit	Base + 0x0090
RUN Peripheral Configuration Registers	ME_RUN_PC5	32-bit	Base + 0x0094
RUN Peripheral Configuration Registers	ME_RUN_PC6	32-bit	Base + 0x0098
RUN Peripheral Configuration Registers	ME_RUN_PC7	32-bit	Base + 0x009C
Low Power Peripheral Configuration Registers	ME_LP_PC0	32-bit	Base + 0x00A0
Low Power Peripheral Configuration Registers	ME_LP_PC1	32-bit	Base + 0x00A4
Low Power Peripheral Configuration Registers	ME_LP_PC2	32-bit	Base + 0x00A8
Low Power Peripheral Configuration Registers	ME_LP_PC3	32-bit	Base + 0x00AC
Low Power Peripheral Configuration Registers	ME_LP_PC4	32-bit	Base + 0x00B0
Low Power Peripheral Configuration Registers	ME_LP_PC5	32-bit	Base + 0x00B4
Low Power Peripheral Configuration Registers	ME_LP_PC6	32-bit	Base + 0x00B8
Low Power Peripheral Configuration Registers	ME_LP_PC7	32-bit	Base + 0x00BC
Reserved	-	-	(Base + 0x00C0) – (Base + 0x00C3)
DSPI0 Control	ME_PCTL4	8-bit	Base + 0x00C4
DSPI1 Control	ME_PCTL5	8-bit	Base + 0x00C5
DSPI2 Control	ME_PCTL6	8-bit	Base + 0x00C6
Reserved	-	-	(Base + 0x00C7) – (Base + 0x00CF)
FlexCAN0 Control	ME_PCTL16	8-bit	Base + 0x00D0
FlexCAN1 Control	ME_PCTL17	8-bit	Base + 0x00D1
FlexCAN2 Control	ME_PCTL18	8-bit	Base + 0x00D2

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
FlexCAN3 Control	ME_PCTL19	8-bit	Base + 0x00D3
FlexCAN4 Control	ME_PCTL20	8-bit	Base + 0x00D4
FlexCAN5 Control	ME_PCTL21	8-bit	Base + 0x00D5
Reserved	-	-	(Base + 0x00D6) – (Base + 0x00DF)
ADC0 Control	ME_PCTL32	8-bit	Base + 0x00E0
Reserved	-	-	(Base + 0x00E1) – (Base + 0x00EB)
I2C0 Control	ME_PCTL44	8-bit	Base + 0x00EC
Reserved	-	-	(Base + 0x00ED) – (Base + 0x00EF)
LINFlex0 Control	ME_PCTL48	8-bit	Base + 0x00F0
LINFlex1 Control	ME_PCTL49	8-bit	Base + 0x00F1
LINFlex2 Control	ME_PCTL50	8-bit	Base + 0x00F2
LINFlex3 Control	ME_PCTL51	8-bit	Base + 0x00F3
Reserved	-	-	(Base + 0x00F4) – (Base + 0x00F8)
CTU Control	ME_PCTL57	8-bit	Base + 0x00F9
Reserved	-	-	(Base + 0x00FA) – (Base + 0x00FB)
CAN Sampler Control	ME_PCTL60	8-bit	Base + 0x00FC
Reserved	-	-	(Base + 0x00FD) – (Base + 0x0103)
SIUL Control	ME_PCTL68	8-bit	Base + 0x0104
WKPU Control	ME_PCTL69	8-bit	Base + 0x0105
Reserved	-	-	(Base + 0x0106) – (Base + 0x0107)
eMIOS0 Control	ME_PCTL72	8-bit	Base + 0x0108
eMIOS1 Control	ME_PCTL73	8-bit	Base + 0x0109
Reserved	-	-	(Base + 0x010A) – (Base + 0x011A)
RTC_API Control	ME_PCTL91	8-bit	Base + 0x011B
PIT Control	ME_PCTL92	8-bit	Base + 0x011C
Reserved	—	—	(Base + 0x011D) – (Base + 0x0127)
CMU Control	ME_PCTL104	8-bit	Base + 0x0128
Reserved	—	—	(Base + 0x0129) – (Base + 0x014F)

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
FXOSC			**0xC3FE_0000**
Fast External Crystal Oscillator Control Register	FXOSC_CTL	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x003F)
SXOSC			**0xC3FE_0040**
Slow External Crystal Oscillator Control Register	SXOSC_CTL	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x005F)
FIRC Digital Interface			**0xC3FE_0060**
RC Digital Interface Registers	RC_CTL	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x007F)
SIRC Digital Interface			**0xC3FE_0080**
Slow Power RC Control Register	LPRC_CTL	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x009F)
FMPLL			**0xC3FE_00A0**
Control Register	PLLD0_CR	32-bit	Base + 0x0000
PLLD Modulation Register	PLLD0_MR	32-bit	Base + 0x0004
Reserved	—	—	(Base + 0x0008) – (Base + 0x00FF)
CMU			**0xC3FE_0100**
Control Status Register	CMU_CSR	32-bit	Base + 0x0000
Frequency Display Register	CMU_FDR	32-bit	Base + 0x0004
High Frequency Reference Register	CMU_HFREFR_A	32-bit	Base + 0x0008
Low Frequency Reference Register	CMU_LFREFR_A	32-bit	Base + 0x000C
Interrupt Status Register	CMU_ISR	32-bit	Base + 0x0010
Reserved	—	—	(Base + 0x0014) – (Base + 0x0017)
Measurement Duration Register	CMU_MDR	32-bit	Base + 0x0018
Reserved	—	—	(Base + 0x001C) – (Base + 0x036F)
Clock Generation Module (MC_CGM)			**0xC3FE_0370**
Output Clock Enable Register	CGM_OC_EN	32-bit	Base + 0x0000
Output Clock Division Select Register	CGM_OCDS_SC	32-bit	Base + 0x0004
System Clock Select Status Register	CGM_SC_SS	32-bit	Base + 0x0008
System Clock Divider Configuration 0 Registers	CGM_SC_DC0	8-bit	Base + 0x000C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
System Clock Divider Configuration 1 Registers	CGM_SC_DC1	8-bit	Base + 0x000D
System Clock Divider Configuration 2 Registers	CGM_SC_DC2	8-bit	Base + 0x000E
Reset Generation Module (MC_RGM)			**0xC3FE_4000**
Functional Event Status	RGM_FES	16-bit	Base + 0x0000
Destructive Event Status	RGM_DES	16-bit	Base + 0x0002
Functional Event Reset Disable	RGM_FERD	16-bit	Base + 0x0004
Destructive Event Reset Disable	RGM_DERD	16-bit	Base + 0x0006
Reserved	—	—	(Base + 0x0008) – (Base + 0x000F)
Functional Event Alternate Request	RGM_FEAR	16-bit	Base + 0x0010
Destructive Event Alternate Request	RGM_DEAR	16-bit	Base + 0x0012
Reserved	—	—	(Base + 0x0014) – (Base + 0x0017)
Functional Event Short Sequence	RGM_FESS	16-bit	Base + 0x0018
STANDBY reset sequence	RGM_STDBY	16-bit	Base + 0x001A
Functional Bidirectional Reset Enable	RGM_FBRE	16-bit	Base + 0x001C
Reserved	—	—	(Base + 0x001E) – (Base + 0x3FFF)
Power Control Unit (MC_PCU)			**0xC3FE_8000**
Power domain #0 configuration register	PCONF0	32-bit	Base + 0x0000
Power domain #1 configuration register	PCONF1	32-bit	Base + 0x0004
Power domain #2 configuration register	PCONF2	32-bit	Base + 0x0008
Reserved	—	—	(Base + 0x000C) – (Base + 0x003F)
Power Domain Status Register	PSTAT	32-bit	Base + 0x0040
Reserved	—	—	(Base + 0x0044) – (Base + 0x007C)
Voltage Regulator Control Register	VCTL	32-bit	Base + 0x0080
Reserved	—	—	(Base + 0x0084) – (Base + 0x3FFF)
Real Time Counter (RTC/API)			**0xC3FE_C000**
RTC Supervisor Control Register	RTCSUPV	32-bit	Base + 0x0000
RTC Control Register	RTCC	32-bit	Base + 0x0004
RTC Status Register	RTCS	32-bit	Base + 0x0008
RTC Counter Register	RTCCNT	32-bit	Base + 0x000C
Reserved	—	—	(Base + 0x0010) – (Base + 0x3FFF)

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Periodic Interrupt Timer (PIT)			**0xC3FF_0000**
PIT Module Control Register	PITMCR	32-bit	Base + 0x0000
Reserved	—	—	Base + (0x0004 – 0x00FC)
Timer Load Value Register	LDVAL0	32-bit	Base + 0x0100
Current Timer Value Register 0	CVAL0	32-bit	Base + 0x0104
Timer Control Register 0	TCTRL0	32-bit	Base + 0x0108
Timer Flag Register 0	TFLG0	32-bit	Base + 0x010C
Timer Load Value Register 1	LDVAL1	32-bit	Base + 0x0110
Current Timer Value Register 1	CVAL1	32-bit	Base + 0x0114
Timer Control Register 1	TCTRL1	32-bit	Base + 0x0118
Timer Flag Register 1	TFLG1	32-bit	Base + 0x011C
Timer Load Value Register 2	LDVAL2	32-bit	Base + 0x0120
Current Timer Value Register 2	CVAL2	32-bit	Base + 0x0124
Timer Control Register 2	TCTRL2	32-bit	Base + 0x0128
Timer Flag Register 2	TFLG2	32-bit	Base + 0x012C
Timer Load Value Register 3	LDVAL3	32-bit	Base + 0x0130
Current Timer Value Register 3	CVAL3	32-bit	Base + 0x0134
Timer Control Register 3	TCTRL3	32-bit	Base + 0x0138
Timer Flag Register 3	TFLG3	32-bit	Base + 0x013C
Timer Load Value Register 4	LDVAL4	32-bit	Base + 0x0140
Current Timer Value Register 4	CVAL4	32-bit	Base + 0x0144
Timer Control Register 4	TCTRL4	32-bit	Base + 0x0148
Timer Flag Register 4	TFLG4	32-bit	Base + 0x014C
Timer Load Value Register 5	LDVAL5	32-bit	Base + 0x0150
Current Timer Value Register 5	CVAL5	32-bit	Base + 0x0154
Timer Control Register 5	TCTRL5	32-bit	Base + 0x0158
Timer Flag Register 5	TFLG5	32-bit	Base + 0x015C
Reserved	—	—	Base + 0x0160 – 0x01FF
ADC			**0xFFE0_0000**
Main Configuration Register	MCR	32-bit	Base + 0x0000
Main Status Register	MSR	32-bit	Base + 0x0004
Reserved	—	—	Base + 0x0008 – 0x000F

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Interrupt Status Register	ISR	32-bit	Base + 0x0010
Channel Pending Register	CEOCFR0	32-bit	Base + 0x0014
Channel Pending Register	CEOCFR1	32-bit	Base + 0x0018
Channel Pending Register	CEOCFR2	32-bit	Base + 0x001C
Interrupt Mask Register	IMR	32-bit	Base + 0x0020
Channel Interrupt Mask Register	CIMR0	32-bit	Base + 0x0024
Channel Interrupt Mask Register	CIMR1	32-bit	Base + 0x0028
Channel Interrupt Mask Register	CIMR2	32-bit	Base + 0x002C
Watchdog Threshold Interrupt Status Register	WTISR	32-bit	Base + 0x0030
Watchdog Threshold Interrupt Mask Register	WTIMR	32-bit	Base + 0x0034
Reserved	—	—	Base + 0x0038 – 0x004F
Threshold Control Register 0	TRC0	32-bit	Base + 0x0050
Threshold Control Register 1	TRC1	32-bit	Base + 0x0054
Threshold Control Register 2	TRC2	32-bit	Base + 0x0058
Threshold Control Register 3	TRC3	32-bit	Base + 0x005C
Threshold Register 0	THRHLR0	32-bit	Base + 0x0060
Threshold Register 1	THRHLR1	32-bit	Base + 0x0064
Threshold Register 2	THRHLR2	32-bit	Base + 0x0068
Threshold Register 3	THRHLR3	32-bit	Base + 0x006C
Presampling Control Register	PSCR	32-bit	Base + 0x0080
Presampling Register 0	PSR0	32-bit	Base + 0x0084
Presampling Register 1	PSR1	32-bit	Base + 0x0088
Presampling Register 2	PSR2	32-bit	Base + 0x008C
Reserved	—	—	Base + 0x0090 – 0x0093
Conversion Timing Register 0	CTR0	32-bit	Base + 0x0094
Conversion Timing Register 1	CTR1	32-bit	Base + 0x0098
Conversion Timing Register 2	CTR2	32-bit	Base + 0x009C
Reserved	—	—	Base + 0x00A0 – 0x00A3
Normal Conversion Mask Register 0	NCMR0	32-bit	Base + 0x00A4
Normal Conversion Mask Register 1	NCMR1	32-bit	Base + 0x00A8
Normal Conversion Mask Register 2	NCMR2	32-bit	Base + 0x00AC

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + 0x00B0 – 0x00B3
Injected Conversion Mask Register 0	JCMR0	32-bit	Base + 0x00B4
Injected Conversion Mask Register 1	JCMR1	32-bit	Base + 0x00B8
Injected Conversion Mask Register 2	JCMR2	32-bit	Base + 0x00BC
Reserved	—	—	Base + 0x00C0 – 0x00C3
Decode Signals Delay Register	DSDR	32-bit	Base + 0x00C4
Power-down Exit Delay Register	PDEDR	32-bit	Base + 0x00C8
Reserved	—	—	Base + 0x00CC – 0x00FF
Channel 0 Data Register	CDR0	32-bit	Base + 0x0100
Channel 1 Data Register	CDR1	32-bit	Base + 0x0104
Channel 2 Data Register	CDR2	32-bit	Base + 0x0108
Channel 3 Data Register	CDR3	32-bit	Base + 0x010C
Channel 4 Data Register	CDR4	32-bit	Base + 0x0110
Channel 5 Data Register	CDR5	32-bit	Base + 0x0114
Channel 6 Data Register	CDR6	32-bit	Base + 0x0118
Channel 7 Data Register	CDR7	32-bit	Base + 0x011C
Channel 8 Data Register	CDR8	32-bit	Base + 0x0120
Channel 9 Data Register	CDR9	32-bit	Base + 0x0124
Channel 10 Data Register	CDR10	32-bit	Base + 0x0128
Channel 11 Data Register	CDR11	32-bit	Base + 0x012C
Channel 12 Data Register	CDR12	32-bit	Base + 0x0130
Channel 13 Data Register	CDR13	32-bit	Base + 0x0134
Channel 14 Data Register	CDR14	32-bit	Base + 0x0138
Channel 15 Data Register	CDR15	32-bit	Base + 0x013C
Reserved	—	—	Base + 0x0140 – 0x017F
Channel 32 Data Register	CDR32	32-bit	Base + 0x0180
Channel 33 Data Register	CDR33	32-bit	Base + 0x0184
Channel 34 Data Register	CDR34	32-bit	Base + 0x0188
Channel 35 Data Register	CDR35	32-bit	Base + 0x018C
Channel 36 Data Register	CDR36	32-bit	Base + 0x0190
Channel 37 Data Register	CDR37	32-bit	Base + 0x0194

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Channel 38 Data Register	CDR38	32-bit	Base + 0x0198
Channel 39 Data Register	CDR39	32-bit	Base + 0x019C
Channel 40 Data Register	CDR40	32-bit	Base + 0x01A0
Channel 41 Data Register	CDR41	32-bit	Base + 0x01A4
Channel 42 Data Register	CDR42	32-bit	Base + 0x01A8
Channel 43 Data Register	CDR43	32-bit	Base + 0x01AC
Channel 44 Data Register	CDR44	32-bit	Base + 0x01B0
Channel 45 Data Register	CDR45	32-bit	Base + 0x01B4
Channel 46 Data Register	CDR46	32-bit	Base + 0x01B8
Channel 47 Data Register	CDR47	32-bit	Base + 0x01BC
Reserved	—	—	Base + 0x01C0 – 0x01FF
Channel 64 Data Register	CDR64	32-bit	Base + 0x0200
Channel 65 Data Register	CDR65	32-bit	Base + 0x0204
Channel 66 Data Register	CDR66	32-bit	Base + 0x0208
Channel 67 Data Register	CDR67	32-bit	Base + 0x020C
Channel 68 Data Register	CDR68	32-bit	Base + 0x0210
Channel 69 Data Register	CDR69	32-bit	Base + 0x0214
Channel 70 Data Register	CDR70	32-bit	Base + 0x0218
Channel 71 Data Register	CDR71	32-bit	Base + 0x021C
Channel 72 Data Register	CDR72	32-bit	Base + 0x0220
Channel 73 Data Register	CDR73	32-bit	Base + 0x0224
Channel 74 Data Register	CDR74	32-bit	Base + 0x0228
Channel 75 Data Register	CDR75	32-bit	Base + 0x022C
Channel 76 Data Register	CDR76	32-bit	Base + 0x0230
Channel 77 Data Register	CDR77	32-bit	Base + 0x0234
Channel 78 Data Register	CDR78	32-bit	Base + 0x0238
Channel 79 Data Register	CDR79	32-bit	Base + 0x023C
Channel 80 Data Register	CDR80	32-bit	Base + 0x0240
Channel 81 Data Register	CDR81	32-bit	Base + 0x0244
Channel 82 Data Register	CDR82	32-bit	Base + 0x0248
Channel 83 Data Register	CDR83	32-bit	Base + 0x024C
Channel 84 Data Register	CDR84	32-bit	Base + 0x0250
Channel 85 Data Register	CDR85	32-bit	Base + 0x0254

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Channel 86 Data Register	CDR86	32-bit	Base + 0x0258
Channel 87 Data Register	CDR87	32-bit	Base + 0x025C
Channel 88 Data Register	CDR88	32-bit	Base + 0x0260
Channel 89 Data Register	CDR89	32-bit	Base + 0x0264
Channel 90 Data Register	CDR90	32-bit	Base + 0x0268
Channel 91 Data Register	CDR91	32-bit	Base + 0x026C
Channel 92 Data Register	CDR92	32-bit	Base + 0x0270
Channel 93 Data Register	CDR93	32-bit	Base + 0x0274
Channel 94 Data Register	CDR94	32-bit	Base + 0x0278
Channel 95 Data Register	CDR95	32-bit	Base + 0x027C
Reserved	—	—	Base + 0x0280 – 0x02FF
I2C			**0xFFE3_0000**
I2C Bus Address Register	IBAD	8-bit	Base + 0x0000
I2C Bus Frequency Divider Register	IBFD	8-bit	Base + 0x0001
I2C Bus Control Register	IBCR	8-bit	Base + 0x0002
I2C Bus Status Register	IBSR	8-bit	Base + 0x0003
I2C Bus Data I/O Register	IBDR	8-bit	Base + 0x0004
I2C Bus Interrupt Configuration Register	IBIC	8-bit	Base + 0x0005
Reserved	—	—	(Base + 0x0006) – (Base + 0xFFFF)
LINFlex_0			**0xFFE4_0000**
LIN control register 1	LINCR1	32-bit	Base + 0x0000
LIN interrupt enable register	LINIER	32-bit	Base + 0x0004
LIN status register	LINSR	32-bit	Base + 0x0008
LIN error status register	LINESR	32-bit	Base + 0x000C
UART mode control register	UARTCR	32-bit	Base + 0x0010
UART mode status register	UARTSR	32-bit	Base + 0x0014
LIN timeout control status register	LINTCSR	32-bit	Base + 0x0018
LIN output compare register	LINOCR	32-bit	Base + 0x001C
LIN timeout control register	LINTOCR	32-bit	Base + 0x0020
LIN fractional baud rate register	LINFBRR	32-bit	Base + 0x0024
LIN integer baud rate register	LINIBRR	32-bit	Base + 0x0028
LIN checksum field register	LINCFR	32-bit	Base + 0x002C
LIN control register 2	LINCR2	32-bit	Base + 0x0030

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Buffer identifier register	BIDR	32-bit	Base + 0x0034
Buffer data register LSB	BDRL	32-bit	Base + 0x0038
Buffer data register MSB	BDRM	32-bit	Base + 0x003C
Identifier filter enable register	IFER	32-bit	Base + 0x0040
Identifier filter match index	IFMI	32-bit	Base + 0x0044
Identifier filter mode register	IFMR	32-bit	Base + 0x0048
Identifier filter control register 0	IFCR0	32-bit	Base + 0x004C
Identifier filter control register 1	IFCR1	32-bit	Base + 0x0050
Identifier filter control register 2	IFCR2	32-bit	Base + 0x0054
Identifier filter control register 3	IFCR3	32-bit	Base + 0x0058
Identifier filter control register 4	IFCR4	32-bit	Base + 0x005C
Identifier filter control register 5	IFCR5	32-bit	Base + 0x0060
Identifier filter control register 6	IFCR6	32-bit	Base + 0x0064
Identifier filter control register 7	IFCR7	32-bit	Base + 0x0068
Identifier filter control register 8	IFCR8	32-bit	Base + 0x006C
Identifier filter control register 9	IFCR9	32-bit	Base + 0x0070
Identifier filter control register 10	IFCR10	32-bit	Base + 0x0074
Identifier filter control register 11	IFCR11	32-bit	Base + 0x0078
Identifier filter control register 12	IFCR12	32-bit	Base + 0x007C
Identifier filter control register 13	IFCR13	32-bit	Base + 0x0080
Identifier filter control register 14	IFCR14	32-bit	Base + 0x0084
Identifier filter control register 15	IFCR15	32-bit	Base + 0x0088
LINFlex_1			**0xFFE4_4000**
LIN control register 1	LINCR1	32-bit	Base + 0x0000
LIN interrupt enable register	LINIER	32-bit	Base + 0x0004
LIN status register	LINSR	32-bit	Base + 0x0008
LIN error status register	LINESR	32-bit	Base + 0x000C
UART mode control register	UARTCR	32-bit	Base + 0x0010
UART mode status register	UARTSR	32-bit	Base + 0x0014
LIN timeout control status register	LINTCSR	32-bit	Base + 0x0018
LIN output compare register	LINOCR	32-bit	Base + 0x001C
LIN timeout control register	LINTOCR	32-bit	Base + 0x0020
LIN fractional baud rate register	LINFBRR	32-bit	Base + 0x0024
LIN integer baud rate register	LINIBRR	32-bit	Base + 0x0028

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
LIN checksum field register	LINCFR	32-bit	Base + 0x002C
LIN control register 2	LINCR2	32-bit	Base + 0x0030
Buffer identifier register	BIDR	32-bit	Base + 0x0034
Buffer data register LSB	BDRL	32-bit	Base + 0x0038
Buffer data register MSB	BDRM	32-bit	Base + 0x003C
Reserved	—	—	(Base + 0x0040)– (Base + 0x7FFF)
LINFlex_2			**0xFFE4_8000**
LIN control register 1	LINCR1	32-bit	Base + 0x0000
LIN interrupt enable register	LINIER	32-bit	Base + 0x0004
LIN status register	LINSR	32-bit	Base + 0x0008
LIN error status register	LINESR	32-bit	Base + 0x000C
UART mode control register	UARTCR	32-bit	Base + 0x0010
UART mode status register	UARTSR	32-bit	Base + 0x0014
LIN timeout control status register	LINTCSR	32-bit	Base + 0x0018
LIN output compare register	LINOCR	32-bit	Base + 0x001C
LIN timeout control register	LINTOCR	32-bit	Base + 0x0020
LIN fractional baud rate register	LINFBRR	32-bit	Base + 0x0024
LIN integer baud rate register	LINIBRR	32-bit	Base + 0x0028
LIN checksum field register	LINCFR	32-bit	Base + 0x002C
LIN control register 2	LINCR2	32-bit	Base + 0x0030
Buffer identifier register	BIDR	32-bit	Base + 0x0034
Buffer data register LSB	BDRL	32-bit	Base + 0x0038
Buffer data register MSB	BDRM	32-bit	Base + 0x003C
Reserved	—	—	(Base + 0x0040)– (Base + 0xBFFF)
LINFlex_3			**0xFFE4_C000**
LIN control register 1	LINCR1	32-bit	Base + 0x0000
LIN interrupt enable register	LINIER	32-bit	Base + 0x0004
LIN status register	LINSR	32-bit	Base + 0x0008
LIN error status register	LINESR	32-bit	Base + 0x000C
UART mode control register	UARTCR	32-bit	Base + 0x0010
UART mode status register	UARTSR	32-bit	Base + 0x0014
LIN timeout control status register	LINTCSR	32-bit	Base + 0x0018
LIN output compare register	LINOCR	32-bit	Base + 0x001C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
LIN timeout control register	LINTOCR	32-bit	Base + 0x0020
LIN fractional baud rate register	LINFBRR	32-bit	Base + 0x0024
LIN integer baud rate register	LINIBRR	32-bit	Base + 0x0028
LIN checksum field register	LINCFR	32-bit	Base + 0x002C
LIN control register 2	LINCR2	32-bit	Base + 0x0030
Buffer identifier register	BIDR	32-bit	Base + 0x0034
Buffer data register LSB	BDRL	32-bit	Base + 0x0038
Buffer data register MSB	BDRM	32-bit	Base + 0x003C
Reserved	—	—	(Base + 0x0040)– (Base + 0x3FFF)
CTU			**0xFFE6_4000**
Reserved	—	—	Base + 0x0000 – Base + 0x002C
Event Configuration Register0	CTU_EVTCFGR0	32-bit	Base + 0x0030
Event Configuration Register1	CTU_EVTCFGR1	32-bit	Base + 0x0034
Event Configuration Register2	CTU_EVTCFGR2	32-bit	Base + 0x0038
Event Configuration Register3	CTU_EVTCFGR3	32-bit	Base + 0x003C
Event Configuration Register4	CTU_EVTCFGR4	32-bit	Base + 0x0040
Event Configuration Register5	CTU_EVTCFGR5	32-bit	Base + 0x0044
Event Configuration Register6	CTU_EVTCFGR6	32-bit	Base + 0x0048
Event Configuration Register7	CTU_EVTCFGR7	32-bit	Base + 0x004C
Event Configuration Register8	CTU_EVTCFGR8	32-bit	Base + 0x0050
Event Configuration Register9	CTU_EVTCFGR9	32-bit	Base + 0x0054
Event Configuration Register10	CTU_EVTCFGR10	32-bit	Base + 0x0058
Event Configuration Register11	CTU_EVTCFGR11	32-bit	Base + 0x005C
Event Configuration Register12	CTU_EVTCFGR12	32-bit	Base + 0x0060
Event Configuration Register13	CTU_EVTCFGR13	32-bit	Base + 0x0064
Event Configuration Register14	CTU_EVTCFGR14	32-bit	Base + 0x0068
Event Configuration Register15	CTU_EVTCFGR15	32-bit	Base + 0x006C
Event Configuration Register16	CTU_EVTCFGR16	32-bit	Base + 0x0070
Event Configuration Register17	CTU_EVTCFGR17	32-bit	Base + 0x0074
Event Configuration Register18	CTU_EVTCFGR18	32-bit	Base + 0x0078
Event Configuration Register19	CTU_EVTCFGR19	32-bit	Base + 0x007C
Event Configuration Register20	CTU_EVTCFGR20	32-bit	Base + 0x0080
Event Configuration Register21	CTU_EVTCFGR21	32-bit	Base + 0x0084

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Event Configuration Register22	CTU_EVTCFGR22	32-bit	Base + 0x0088
Event Configuration Register23	CTU_EVTCFGR23	32-bit	Base + 0x008C
Event Configuration Register24	CTU_EVTCFGR24	32-bit	Base + 0x0090
Event Configuration Register25	CTU_EVTCFGR25	32-bit	Base + 0x0094
Event Configuration Register26	CTU_EVTCFGR26	32-bit	Base + 0x0098
Event Configuration Register27	CTU_EVTCFGR27	32-bit	Base + 0x009C
Event Configuration Register28	CTU_EVTCFGR28	32-bit	Base + 0x00A0
Event Configuration Register29	CTU_EVTCFGR29	32-bit	Base + 0x00A4
Event Configuration Register30	CTU_EVTCFGR30	32-bit	Base + 0x00A8
Event Configuration Register31	CTU_EVTCFGR31	32-bit	Base + 0x00AC
Event Configuration Register32	CTU_EVTCFGR32	32-bit	Base + 0x00B0
Event Configuration Register33	CTU_EVTCFGR33	32-bit	Base + 0x00B4
Event Configuration Register34	CTU_EVTCFGR34	32-bit	Base + 0x00B8
Event Configuration Register35	CTU_EVTCFGR35	32-bit	Base + 0x00BC
Event Configuration Register36	CTU_EVTCFGR36	32-bit	Base + 0x00C0
Event Configuration Register37	CTU_EVTCFGR37	32-bit	Base + 0x00C4
Event Configuration Register38	CTU_EVTCFGR38	32-bit	Base + 0x00C8
Event Configuration Register39	CTU_EVTCFGR39	32-bit	Base + 0x00CC
Event Configuration Register40	CTU_EVTCFGR40	32-bit	Base + 0x00D0
Event Configuration Register41	CTU_EVTCFGR41	32-bit	Base + 0x00D4
Event Configuration Register42	CTU_EVTCFGR42	32-bit	Base + 0x00D8
Event Configuration Register43	CTU_EVTCFGR43	32-bit	Base + 0x00DC
Event Configuration Register44	CTU_EVTCFGR44	32-bit	Base + 0x00E0
Event Configuration Register45	CTU_EVTCFGR45	32-bit	Base + 0x00E4
Event Configuration Register46	CTU_EVTCFGR46	32-bit	Base + 0x00E8
Event Configuration Register47	CTU_EVTCFGR47	32-bit	Base + 0x00EC
Event Configuration Register48	CTU_EVTCFGR48	32-bit	Base + 0x00F0
Event Configuration Register49	CTU_EVTCFGR49	32-bit	Base + 0x00F4
Event Configuration Register50	CTU_EVTCFGR50	32-bit	Base + 0x00F8
Event Configuration Register51	CTU_EVTCFGR51	32-bit	Base + 0x00FC
Event Configuration Register52	CTU_EVTCFGR52	32-bit	Base + 0x0100
Event Configuration Register53	CTU_EVTCFGR53	32-bit	Base + 0x0104
Event Configuration Register54	CTU_EVTCFGR54	32-bit	Base + 0x0108
Event Configuration Register55	CTU_EVTCFGR55	32-bit	Base + 0x010C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Event Configuration Register56	CTU_EVTCFGR56	32-bit	Base + 0x0110
Event Configuration Register57	CTU_EVTCFGR57	32-bit	Base + 0x0114
Event Configuration Register58	CTU_EVTCFGR58	32-bit	Base + 0x0118
Event Configuration Register59	CTU_EVTCFGR59	32-bit	Base + 0x011C
Event Configuration Register60	CTU_EVTCFGR60	32-bit	Base + 0x0120
Event Configuration Register61	CTU_EVTCFGR61	32-bit	Base + 0x0124
Event Configuration Register62	CTU_EVTCFGR62	32-bit	Base + 0x0128
Event Configuration Register63	CTU_EVTCFGR63	32-bit	Base + 0x012C
Reserved	—	—	(Base + 0x0130) – 0xFFE6_FFFF
CAN Sampler			**0xFFE7_0000**
Control Status Register	CANS_CR	32-bit	Base + 0x0000
Sample Register 0	CAN_SR0	32-bit	Base + 0x0004
Sample Register 1	CAN_SR1	32-bit	Base + 0x0008
Sample Register 2	CAN_SR2	32-bit	Base + 0x000C
Sample Register 3	CAN_SR3	32-bit	Base + 0x0010
Sample Register 4	CAN_SR4	32-bit	Base + 0x0014
Sample Register 5	CAN_SR5	32-bit	Base + 0x0018
Sample Register 6	CAN_SR6	32-bit	Base + 0x001C
Sample Register 7	CAN_SR7	32-bit	Base + 0x0020
Sample Register 8	CAN_SR8	32-bit	Base + 0x0024
Sample Register 9	CAN_SR9	32-bit	Base + 0x0028
Sample Register 10	CAN_SR10	32-bit	Base + 0x002C
Sample Register 11	CAN_SR11	32-bit	Base + 0x0030
Reserved	—	—	(Base + 0x0034) – 0xFFF0_FFFF
MPU			**0xFFF1_0000**
MPU Control/Error Status Register	MPU_CESR	32-bit	Base + 0x0000
Reserved	—	—	Base + 0x0004 – Base + 0x000F
MPU Error Address Register, Slave Port 0	MPU_EAR0	32-bit	Base + 0x0010
MPU Error Detail Register, Slave Port 0	MPU_EDR0	32-bit	Base + 0x0014
MPU Error Address Register, Slave Port 1	MPU_EAR1	32-bit	Base + 0x0018
MPU Error Detail Register, Slave Port 1	MPU_EDR1	32-bit	Base + 0x001c
MPU Error Address Register, Slave Port 2	MPU_EAR2	32-bit	Base + 0x0020

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
MPU Error Detail Register, Slave Port 2	MPU_EDR2	32-bit	Base + 0x0024
MPU Error Address Register, Slave Port 3	MPU_EAR3	32-bit	Base + 0x0028
MPU Error Detail Register, Slave Port 3	MPU_EDR3	32-bit	Base + 0x002C
Reserved	—	—	Base + 0x0030 – Base + 0x03FF
MPU Region Descriptor 0	MPU_RGD0	128	Base + 0x0400
MPU Region Descriptor 1	MPU_RGD1	128	Base + 0x0410
MPU Region Descriptor 2	MPU_RGD2	128	Base + 0x0420
MPU Region Descriptor 3	MPU_RGD3	128	Base + 0x0430
MPU Region Descriptor 4	MPU_RGD4	128	Base + 0x0440
MPU Region Descriptor 5	MPU_RGD5	128	Base + 0x0450
MPU Region Descriptor 6	MPU_RGD6	128	Base + 0x0460
MPU Region Descriptor 7	MPU_RGD7	128	Base + 0x0470
Reserved	—	—	Base + 0x0480 – Base + 0x07FF
MPU RGD Alternate Access Control 0	MPU_RGDAAC0	32-bit	Base + 0x0800
MPU RGD Alternate Access Control 1	MPU_RGDAAC1	32-bit	Base + 0x0804
MPU RGD Alternate Access Control 2	MPU_RGDAAC2	32-bit	Base + 0x0808
MPU RGD Alternate Access Control 3	MPU_RGDAAC3	32-bit	Base + 0x080C
MPU RGD Alternate Access Control 4	MPU_RGDAAC4	32-bit	Base + 0x0810
MPU RGD Alternate Access Control 5	MPU_RGDAAC5	32-bit	Base + 0x0814
MPU RGD Alternate Access Control 6	MPU_RGDAAC6	32-bit	Base + 0x0818
MPU RGD Alternate Access Control 7	MPU_RGDAAC7	32-bit	Base + 0x081C
Reserved	—	—	Base + 0x0820 – Base + 0x3FFF
SWT			**0xFFF3_8000**
Control Register	SWT_CR	32-bit	Base + 0x0000
SWT Interrupt Register	SWT_IR	32-bit	Base + 0x0004
SWT Time-Out Register	SWT_TO	32-bit	Base + 0x0008
SWT Window Register	SWT_WN	32-bit	Base + 0x000C
SWT Service Register	SWT_SR	32-bit	Base + 0x0010
SWT Counter Output Register	SWT_CO	32-bit	Base + 0x0014
Reserved	—	—	(Base + 0x0018) – 0xFFF3_BFFF
STM			**0xFFF3_C000**

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Control Register	STM_CR	32-bit	Base + 0x0000
STM Count Register	STM_CNT	32-bit	Base + 0x0004
Reserved	—	—	Base + (0x0008 – 0x000F)
STM Channel 0 Control Register	STM_CCR0	32-bit	Base + 0x00010
STM Channel 0 Interrupt Register	STM_CIR0	32-bit	Base + 0x00014
STM Channel 0 Compare Register	STM_CMP0	32-bit	Base + 0x00018
Reserved	—	—	Base + (0x001C – 0x001F)
STM Channel 1 Control Register	STM_CCR1	32-bit	Base + 0x00020
STM Channel 1 Interrupt Register	STM_CIR1	32-bit	Base + 0x00024
STM Channel 1 Compare Register	STM_CMP1	32-bit	Base + 0x00028
Reserved	—	—	Base + (0x002C – 0x002F)
STM Channel 2 Control Register	STM_CCR2	32-bit	Base + 0x00030
STM Channel 2 Interrupt Register	STM_CIR2	32-bit	Base + 0x00034
STM Channel 2 Compare Register	STM_CMP2	32-bit	Base + 0x00038
Reserved	—	—	Base + (0x003C – 0x003F)
STM Channel 3 Control Register	STM_CCR3	32-bit	Base + 0x00040
STM Channel 3 Interrupt Register	STM_CIR3	32-bit	Base + 0x00044
STM Channel 3 Compare Register	STM_CMP3	32-bit	Base + 0x00048
Reserved	—	—	Base + (0x003C – 0x03FFF)
ECSM			**0xFFF4_0000**
Processor Core Type	ECSM_PCT	16-bit	Base + 0x0000
SOC-Defined Platform Revision	ECSM_REV	16-bit	Base + 0x0002
Reserved	—	—	Base + (0x0004 – 0x0007)
IPS On-Platform Module Configuration	ECSM_IMC	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x0012)
Miscellaneous Wakeup Control Register	ECSM_MWCR	8-bit	Base + 0x0013
Reserved	—	—	Base + (0x0014 – 0x001E)
Miscellaneous Interrupt Register	ECSM_MIR	8-bit	Base + 0x001F

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + (0x0020 – 0x0023)
Miscellaneous User Defined Control Register	ECSM_MUDCR	32-bit	Base + 0x0024
Reserved	—	—	Base + (0x0028 – 0x0042)
ECC Configuration Register	ECSM_ECR	8-bit	Base + 0x0043
Reserved	—	—	Base + (0x0044 – 0x0046)
ECC Status Register	ECSM_ESR	8-bit	Base + 0x0047
Reserved	—	—	Base + (0x0048 – 0x0049)
ECC Error Generation Register	ECSM_EEGR	16-bit	Base + 0x004A
Reserved	—	—	Base + (0x04C – 0x004F)
Platform Flash ECC Error Address Register	ECSM_PFEAR	32-bit	Base + 0x0050
Reserved	—	—	Base + (0x054 – 0x0055)
Platform Flash ECC Master Number Register	ECSM_PFEMR	8-bit	Base + 0x0056
Platform Flash ECC Attributes Register	ECSM_PFEAT	8-bit	Base + 0x0057
Reserved	—	—	Base + (0x058 – 0x005B)
Platform Flash ECC Data Register	ECSM_PFEDR	32-bit	Base + 0x005C
Platform RAM ECC Address Register	ECSM_PREAR	32-bit	Base + 0x0060
Reserved	—	—	Base + 0x064
Platform RAM ECC Syndrome Register	ECSM_PRESR	8-bit	Base + 0x0065
Platform RAM ECC Master Number Register	ECSM_PREMR	8-bit	Base + 0x0066
Platform RAM ECC Attributes Register	ECSM_PREAT	8-bit	Base + 0x0067
Reserved	—	—	Base + (0x068 – 0x006B)
Platform RAM ECC Data Register	ECSM_PREDR	32-bit	Base + 0x006C
Reserved	—	—	Base + (0x0070 – 0x3FFF)
INTC			**0xFFF4_8000**
Block Configuration Register	INTC_PBCR	32-bit	Base + 0x0000
Reserved	—	—	Base + (0x0004 – 0x0007)
Current Priority Register	INTC_CPR	32-bit	Base + 0x0008

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + (0x000C – 0x000F)
Interrupt Acknowledge Register	INTC_IACKR	32-bit	Base + 0x0010
Reserved	—	—	Base + (0x0014 – 0x0017)
End of Interrupt Register	INTC_EOIR	32-bit	Base + 0x0018
Reserved	—	—	Base + (0x001C – 0x001F)
Software Set/Clear Interrupt Register	INTC_SSCIR0_3	32-bit	Base + 0x0020
Software Set/Clear Interrupt Register	INTC_SSCIR4_7	32-bit	Base + 0x0024
Reserved	—	—	Base + (0x0028 – 0x003F)
Priority Select Register	INTC_PSR0_3	32-bit	Base + 0x0040
Priority Select Register	INTC_PSR4_7	32-bit	Base + 0x0044
Priority Select Register	INTC_PSR8_11	32-bit	Base + 0x0048
Priority Select Register	INTC_PSR12_15	32-bit	Base + 0x004C
Priority Select Register	INTC_PSR16_19	32-bit	Base + 0x0050
Priority Select Register	INTC_PSR20_23	32-bit	Base + 0x0054
Priority Select Register	INTC_PSR24_27	32-bit	Base + 0x0058
Priority Select Register	INTC_PSR28_31	32-bit	Base + 0x005C
Priority Select Register	INTC_PSR32_35	32-bit	Base + 0x0060
Priority Select Register	INTC_PSR36_39	32-bit	Base + 0x0064
Priority Select Register	INTC_PSR40_43	32-bit	Base + 0x0068
Priority Select Register	INTC_PSR44_47	32-bit	Base + 0x006C
Priority Select Register	INTC_PSR48_51	32-bit	Base + 0x0070
Priority Select Register	INTC_PSR52_55	32-bit	Base + 0x0074
Priority Select Register	INTC_PSR56_59	32-bit	Base + 0x0078
Priority Select Register	INTC_PSR60_63	32-bit	Base + 0x007C
Priority Select Register	INTC_PSR64_67	32-bit	Base + 0x0080
Priority Select Register	INTC_PSR68_71	32-bit	Base + 0x0084
Priority Select Register	INTC_PSR72_75	32-bit	Base + 0x0088
Priority Select Register	INTC_PSR76_79	32-bit	Base + 0x008C
Priority Select Register	INTC_PSR80_83	32-bit	Base + 0x0090
Priority Select Register	INTC_PSR84_87	32-bit	Base + 0x0094
Priority Select Register	INTC_PSR88_91	32-bit	Base + 0x0098

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Priority Select Register	INTC_PSR92_95	32-bit	Base + 0x009C
Priority Select Register	INTC_PSR96_99	32-bit	Base + 0x00A0
Priority Select Register	INTC_PSR100_103	32-bit	Base + 0x00A4
Priority Select Register	INTC_PSR104_107	32-bit	Base + 0x00A8
Priority Select Register	INTC_PSR108_111	32-bit	Base + 0x00AC
Priority Select Register	INTC_PSR112_115	32-bit	Base + 0x00B0
Priority Select Register	INTC_PSR116_119	32-bit	Base + 0x00B4
Priority Select Register	INTC_PSR120_123	32-bit	Base + 0x00B8
Priority Select Register	INTC_PSR124_127	32-bit	Base + 0x00BC
Priority Select Register	INTC_PSR128_131	32-bit	Base + 0x00C0
Priority Select Register	INTC_PSR132_135	32-bit	Base + 0x00C4
Priority Select Register	INTC_PSR136_139	32-bit	Base + 0x00C8
Priority Select Register	INTC_PSR140_143	32-bit	Base + 0x00CC
Priority Select Register	INTC_PSR144_147	32-bit	Base + 0x00D0
Priority Select Register	INTC_PSR148_151	32-bit	Base + 0x00D4
Priority Select Register	INTC_PSR152_155	32-bit	Base + 0x00D8
Priority Select Register	INTC_PSR156_159	32-bit	Base + 0x00DC
Priority Select Register	INTC_PSR160_163	32-bit	Base + 0x00E0
Priority Select Register	INTC_PSR164_167	32-bit	Base + 0x00E4
Priority Select Register	INTC_PSR168_171	32-bit	Base + 0x00E8
Priority Select Register	INTC_PSR172_175	32-bit	Base + 0x00EC
Priority Select Register	INTC_PSR176_179	32-bit	Base + 0x00F0
Priority Select Register	INTC_PSR180_183	32-bit	Base + 0x00F4
Priority Select Register	INTC_PSR184_187	32-bit	Base + 0x00F8
Priority Select Register	INTC_PSR188_191	32-bit	Base + 0x00FC
Priority Select Register	INTC_PSR192_195	32-bit	Base + 0x0100
Priority Select Register	INTC_PSR196_199	32-bit	Base + 0x0104
Priority Select Register	INTC_PSR200_203	32-bit	Base + 0x0108
Priority Select Register	INTC_PSR204_207	32-bit	Base + 0x010C
Priority Select Register	INTC_PSR208_210	32-bit	Base + 0x0110
DSPI_0		**0xFFF9_0000**	
Module Configuration Register	PMCR	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x0007)

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Transfer Count Register	TCR	32-bit	Base + 0x0008
Clock and Transfer Attribute Registers	CTAR0	32-bit	Base + 0x000C
Clock and Transfer Attribute Registers	CTAR1	32-bit	Base + 0x0010
Clock and Transfer Attribute Registers	CTAR2	32-bit	Base + 0x0014
Clock and Transfer Attribute Registers	CTAR3	32-bit	Base + 0x0018
Clock and Transfer Attribute Registers	CTAR4	32-bit	Base + 0x001C
Clock and Transfer Attribute Registers	CTAR5	32-bit	Base + 0x0020
Reserved	—	—	(Base + 0x0024) – (Base + 0x0028)
Status Register	SR	32-bit	Base + 0x002C
DSPI Interrupt Request Enable Register	RSER	32-bit	Base + 0x0030
PUSH TX FIFO Register	PUSHR	32-bit	Base + 0x0034
POP RX FIFO Register	POPR	32-bit	Base + 0x0038
DSPI Transmit FIFO Registers	TXFR0	32-bit	Base + 0x003C
DSPI Transmit FIFO Registers	TXFR1	32-bit	Base + 0x0040
DSPI Transmit FIFO Registers	TXFR2	32-bit	Base + 0x0044
DSPI Transmit FIFO Registers	TXFR3	32-bit	Base + 0x0048
Reserved	—	—	(Base + 0x004C) – (Base + 0x007B)
Receive FIFO Registers	RXFR0	32-bit	Base + 0x007C
Receive FIFO Registers	RXFR1	32-bit	Base + 0x0080
Receive FIFO Registers	RXFR2	32-bit	Base + 0x0084
Receive FIFO Registers	RXFR3	32-bit	Base + 0x0088
Reserved	—	—	(Base + 0x008C) – (Base + 0x3FFF)
DSPI_1		**0xFFF9_4000**	
Module Configuration Register	PMCR	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x0007)
Transfer Count Register	TCR	32-bit	Base + 0x0008
Clock and Transfer Attribute Registers	CTAR0	32-bit	Base + 0x000C
Clock and Transfer Attribute Registers	CTAR1	32-bit	Base + 0x0010
Clock and Transfer Attribute Registers	CTAR2	32-bit	Base + 0x0014
Clock and Transfer Attribute Registers	CTAR3	32-bit	Base + 0x0018
Clock and Transfer Attribute Registers	CTAR4	32-bit	Base + 0x001C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Clock and Transfer Attribute Registers	CTAR5	32-bit	Base + 0x0020
Clock and Transfer Attribute Registers	CTAR6	32-bit	Base + 0x0024
Clock and Transfer Attribute Registers	CTAR7	32-bit	Base + 0x0028
Status Register	SR	32-bit	Base + 0x002C
DSPI Interrupt Request Enable Register	RSER	32-bit	Base + 0x0030
PUSH TX FIFO Register	PUSHR	32-bit	Base + 0x0034
POP RX FIFO Register	POPR	32-bit	Base + 0x0038
DSPI Transmit FIFO Registers	TXFR0	32-bit	Base + 0x003C
DSPI Transmit FIFO Registers	TXFR1	32-bit	Base + 0x0040
DSPI Transmit FIFO Registers	TXFR2	32-bit	Base + 0x0044
DSPI Transmit FIFO Registers	TXFR3	32-bit	Base + 0x0048
Reserved	—	—	(Base + 0x004C) – (Base + 0x007B)
Receive FIFO Registers	RXFR0	32-bit	Base + 0x007C
Receive FIFO Registers	RXFR1	32-bit	Base + 0x0080
Receive FIFO Registers	RXFR2	32-bit	Base + 0x0084
Receive FIFO Registers	RXFR3	32-bit	Base + 0x0088
Reserved	—	—	(Base + 0x0090) – (Base + 0x3FFF)
DSPI_2		**0xFFF9_8000**	
Module Configuration Register	PMCR	32-bit	Base + 0x0000
Reserved	—	—	(Base + 0x0004) – (Base + 0x0007)
Transfer Count Register	TCR	32-bit	Base + 0x0008
Clock and Transfer Attribute Registers	CTAR0	32-bit	Base + 0x000C
Clock and Transfer Attribute Registers	CTAR1	32-bit	Base + 0x0010
Clock and Transfer Attribute Registers	CTAR2	32-bit	Base + 0x0014
Clock and Transfer Attribute Registers	CTAR3	32-bit	Base + 0x0018
Clock and Transfer Attribute Registers	CTAR4	32-bit	Base + 0x001C
Clock and Transfer Attribute Registers	CTAR5	32-bit	Base + 0x0020
Clock and Transfer Attribute Registers	CTAR6	32-bit	Base + 0x0024
Clock and Transfer Attribute Registers	CTAR7	32-bit	Base + 0x0028
Status Register	SR	32-bit	Base + 0x002C
DSPI Interrupt Request Enable Register	RSER	32-bit	Base + 0x0030
PUSH TX FIFO Register	PUSHR	32-bit	Base + 0x0034

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
POP RX FIFO Register	POPR	32-bit	Base + 0x0038
DSPI Transmit FIFO Registers	TXFR0	32-bit	Base + 0x003C
DSPI Transmit FIFO Registers	TXFR1	32-bit	Base + 0x0040
DSPI Transmit FIFO Registers	TXFR2	32-bit	Base + 0x0044
DSPI Transmit FIFO Registers	TXFR3	32-bit	Base + 0x0048
Reserved	—	—	(Base + 0x004C) – (Base + 0x007B)
Receive FIFO Registers	RXFR0	32-bit	Base + 0x007C
Receive FIFO Registers	RXFR1	32-bit	Base + 0x0080
Receive FIFO Registers	RXFR2	32-bit	Base + 0x0084
Receive FIFO Registers	RXFR3	32-bit	Base + 0x0088
Reserved	—	—	(Base + 0x0090) – (0xFFFF_BFFF)
FlexCAN_0		**0xFFFC_0000**	
Module Configuration Register	MCR	32-bit	Base + 0x0000
Control Register	CTRL	32-bit	Base + 0x0004
Free Running Timer	TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x000F)
Rx Global Mask Register	RXGMASK	32-bit	Base + 0x0010
Rx 14 Mask Register	RX14MASK	32-bit	Base + 0x0014
Rx 15 Mask Register	RX15MASK	32-bit	Base + 0x0018
Error Counter Register	ECR	32-bit	Base + 0x001C
Error and Status Register	ESR	32-bit	Base + 0x0020
Interrupt Masks 2 Register	IMASK2	32-bit	Base + 0x0024
Interrupt Masks 1 Register	IMASK1	32-bit	Base + 0x0028
Interrupt Flags 2 Register	IFLAG2	32-bit	Base + 0x002C
Interrupt Flags 1 Register	IFLAG1	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034 – 0x007F)
Message Buffer 0	MB0	128 bits per MB	Base + 0x0080
Message Buffer 1	MB1	128 bits per MB	Base + 0x0090
Message Buffer 2	MB2	128 bits per MB	Base + 0x00A0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 3	MB3	128 bits per MB	Base + 0x00B0
Message Buffer 4	MB4	128 bits per MB	Base + 0x00C0
Message Buffer 5	MB5	128 bits per MB	Base + 0x00D0
Message Buffer 6	MB6	128 bits per MB	Base + 0x00E0
Message Buffer 7	MB7	128 bits per MB	Base + 0x00F0
Message Buffer 8	MB8	128 bits per MB	Base + 0x0100
Message Buffer 9	MB9	128 bits per MB	Base + 0x0110
Message Buffer 10	MB10	128 bits per MB	Base + 0x0120
Message Buffer 11	MB11	128 bits per MB	Base + 0x0130
Message Buffer 12	MB12	128 bits per MB	Base + 0x0140
Message Buffer 13	MB13	128 bits per MB	Base + 0x0150
Message Buffer 14	MB14	128 bits per MB	Base + 0x0160
Message Buffer 15	MB15	128 bits per MB	Base + 0x0170
Message Buffer 16	MB16	128 bits per MB	Base + 0x0180
Message Buffer 17	MB17	128 bits per MB	Base + 0x0190
Message Buffer 18	MB18	128 bits per MB	Base + 0x01A0
Message Buffer 19	MB19	128 bits per MB	Base + 0x01B0
Message Buffer 20	MB20	128 bits per MB	Base + 0x01C0
Message Buffer 21	MB21	128 bits per MB	Base + 0x01D0
Message Buffer 22	MB22	128 bits per MB	Base + 0x01E0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 23	MB23	128 bits per MB	Base + 0x01F0
Message Buffer 24	MB24	128 bits per MB	Base + 0x0200
Message Buffer 25	MB25	128 bits per MB	Base + 0x0210
Message Buffer 26	MB26	128 bits per MB	Base + 0x0220
Message Buffer 27	MB27	128 bits per MB	Base + 0x0230
Message Buffer 28	MB28	128 bits per MB	Base + 0x0240
Message Buffer 29	MB29	128 bits per MB	Base + 0x0250
Message Buffer 30	MB30	128 bits per MB	Base + 0x0260
Message Buffer 31	MB31	128 bits per MB	Base + 0x0270
Message Buffer 32	MB32	128 bits per MB	Base + 0x0280
Message Buffer 33	MB33	128 bits per MB	Base + 0x0290
Message Buffer 34	MB34	128 bits per MB	Base + 0x02A0
Message Buffer 35	MB35	128 bits per MB	Base + 0x02B0
Message Buffer 36	MB36	128 bits per MB	Base + 0x02C0
Message Buffer 37	MB37	128 bits per MB	Base + 0x02D0
Message Buffer 38	MB38	128 bits per MB	Base + 0x02E0
Message Buffer 39	MB39	128 bits per MB	Base + 0x02F0
Message Buffer 40	MB40	128 bits per MB	Base + 0x0300
Message Buffer 41	MB41	128 bits per MB	Base + 0x0310
Message Buffer 42	MB42	128 bits per MB	Base + 0x0320

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 43	MB43	128 bits per MB	Base + 0x0330
Message Buffer 44	MB44	128 bits per MB	Base + 0x0340
Message Buffer 45	MB45	128 bits per MB	Base + 0x0350
Message Buffer 46	MB46	128 bits per MB	Base + 0x0360
Message Buffer 47	MB47	128 bits per MB	Base + 0x0370
Message Buffer 48	MB48	128 bits per MB	Base + 0x0380
Message Buffer 49	MB49	128 bits per MB	Base + 0x0390
Message Buffer 50	MB50	128 bits per MB	Base + 0x03A0
Message Buffer 51	MB51	128 bits per MB	Base + 0x03B0
Message Buffer 52	MB52	128 bits per MB	Base + 0x03C0
Message Buffer 53	MB53	128 bits per MB	Base + 0x03D0
Message Buffer 54	MB54	128 bits per MB	Base + 0x03E0
Message Buffer 55	MB55	128 bits per MB	Base + 0x03F0
Message Buffer 56	MB56	128 bits per MB	Base + 0x0400
Message Buffer 57	MB57	128 bits per MB	Base + 0x0410
Message Buffer 58	MB58	128 bits per MB	Base + 0x0420
Message Buffer 59	MB59	128 bits per MB	Base + 0x0430
Message Buffer 60	MB60	128 bits per MB	Base + 0x0440
Message Buffer 61	MB61	128 bits per MB	Base + 0x0450
Message Buffer 62	MB62	128 bits per MB	Base + 0x0460

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 63	MB63	128 bits per MB	Base + 0x0470
Reserved	—	—	(Base + 0x0480) – (Base + 0x087F)
RX Individual Mask Register 0	RXIMR0	32-bit	Base + 0x0880
RX Individual Mask Register 1	RXIMR1	32-bit	Base + 0x0884
RX Individual Mask Register 2	RXIMR2	32-bit	Base + 0x0888
RX Individual Mask Register 3	RXIMR3	32-bit	Base + 0x088C
RX Individual Mask Register 4	RXIMR4	32-bit	Base + 0x0890
RX Individual Mask Register 5	RXIMR5	32-bit	Base + 0x0894
RX Individual Mask Register 6	RXIMR6	32-bit	Base + 0x0898
RX Individual Mask Register 7	RXIMR7	32-bit	Base + 0x089C
RX Individual Mask Register 8	RXIMR8	32-bit	Base + 0x08A0
RX Individual Mask Register 9	RXIMR9	32-bit	Base + 0x08A4
RX Individual Mask Register 10	RXIMR10	32-bit	Base + 0x08A8
RX Individual Mask Register 11	RXIMR11	32-bit	Base + 0x08AC
RX Individual Mask Register 12	RXIMR12	32-bit	Base + 0x08B0
RX Individual Mask Register 13	RXIMR13	32-bit	Base + 0x08B4
RX Individual Mask Register 14	RXIMR14	32-bit	Base + 0x08B8
RX Individual Mask Register 15	RXIMR15	32-bit	Base + 0x08BC
RX Individual Mask Register 16	RXIMR16	32-bit	Base + 0x08C0
RX Individual Mask Register 17	RXIMR17	32-bit	Base + 0x08C4
RX Individual Mask Register 18	RXIMR18	32-bit	Base + 0x08C8
RX Individual Mask Register 19	RXIMR19	32-bit	Base + 0x08CC
RX Individual Mask Register 20	RXIMR20	32-bit	Base + 0x08D0
RX Individual Mask Register 21	RXIMR21	32-bit	Base + 0x08D4
RX Individual Mask Register 22	RXIMR22	32-bit	Base + 0x08D8
RX Individual Mask Register 23	RXIMR23	32-bit	Base + 0x08DC
RX Individual Mask Register 24	RXIMR24	32-bit	Base + 0x08E0
RX Individual Mask Register 25	RXIMR25	32-bit	Base + 0x08E4
RX Individual Mask Register 26	RXIMR26	32-bit	Base + 0x08E8
RX Individual Mask Register 27	RXIMR27	32-bit	Base + 0x08EC
RX Individual Mask Register 28	RXIMR28	32-bit	Base + 0x08F0
RX Individual Mask Register 29	RXIMR29	32-bit	Base + 0x08F4
RX Individual Mask Register 30	RXIMR30	32-bit	Base + 0x08F8

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 31	RXIMR31	32-bit	Base + 0x08FC
RX Individual Mask Register 32	RXIMR32	32-bit	Base + 0x0900
RX Individual Mask Register 33	RXIMR33	32-bit	Base + 0x0904
RX Individual Mask Register 34	RXIMR34	32-bit	Base + 0x0908
RX Individual Mask Register 35	RXIMR35	32-bit	Base + 0x090C
RX Individual Mask Register 36	RXIMR36	32-bit	Base + 0x0910
RX Individual Mask Register 37	RXIMR37	32-bit	Base + 0x0914
RX Individual Mask Register 38	RXIMR38	32-bit	Base + 0x0918
RX Individual Mask Register 39	RXIMR39	32-bit	Base + 0x091C
RX Individual Mask Register 40	RXIMR40	32-bit	Base + 0x0920
RX Individual Mask Register 41	RXIMR41	32-bit	Base + 0x0924
RX Individual Mask Register 42	RXIMR42	32-bit	Base + 0x0928
RX Individual Mask Register 43	RXIMR43	32-bit	Base + 0x092C
RX Individual Mask Register 44	RXIMR44	32-bit	Base + 0x0930
RX Individual Mask Register 45	RXIMR45	32-bit	Base + 0x0934
RX Individual Mask Register 46	RXIMR46	32-bit	Base + 0x0938
RX Individual Mask Register 47	RXIMR47	32-bit	Base + 0x093C
RX Individual Mask Register 48	RXIMR48	32-bit	Base + 0x0940
RX Individual Mask Register 49	RXIMR49	32-bit	Base + 0x0944
RX Individual Mask Register 50	RXIMR50	32-bit	Base + 0x0948
RX Individual Mask Register 51	RXIMR51	32-bit	Base + 0x094C
RX Individual Mask Register 52	RXIMR52	32-bit	Base + 0x0950
RX Individual Mask Register 53	RXIMR53	32-bit	Base + 0x0954
RX Individual Mask Register 54	RXIMR54	32-bit	Base + 0x0958
RX Individual Mask Register 55	RXIMR55	32-bit	Base + 0x095C
RX Individual Mask Register 56	RXIMR56	32-bit	Base + 0x0960
RX Individual Mask Register 57	RXIMR57	32-bit	Base + 0x0964
RX Individual Mask Register 58	RXIMR58	32-bit	Base + 0x0968
RX Individual Mask Register 59	RXIMR59	32-bit	Base + 0x096C
RX Individual Mask Register 60	RXIMR60	32-bit	Base + 0x0970
RX Individual Mask Register 61	RXIMR61	32-bit	Base + 0x0974
RX Individual Mask Register 62	RXIMR62	32-bit	Base + 0x0978
RX Individual Mask Register 63	RXIMR63	32-bit	Base + 0x097C

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	(Base + 0x0980) – (Base + 0x3FFF)
FlexCAN_1		**0xFFFC_4000**	
Module Configuration	MCR	32-bit	Base + 0x0000
Control Register	CTRL	32-bit	Base + 0x0004
Free Running Timer	TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x000F)
Rx Global Mask Register	RXGMASK	32-bit	Base + 0x0010
Rx 14 Mask Register	RX14MASK	32-bit	Base + 0x0014
Rx 15 Mask Register	RX15MASK	32-bit	Base + 0x0018
Error Counter Register	ECR	32-bit	Base + 0x001C
Error and Status Register	ESR	32-bit	Base + 0x0020
Interrupt Masks 2 Register	IMASK2	32-bit	Base + 0x0024
Interrupt Masks 1 Register	IMASK1	32-bit	Base + 0x0028
Interrupt Flags 2 Register	IFLAG2	32-bit	Base + 0x002C
Interrupt Flags 1 Register	IFLAG1	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034 – 0x007F)
Message Buffer 0	MB0	128 bits per MB	Base + 0x0080
Message Buffer 1	MB1	128 bits per MB	Base + 0x0090
Message Buffer 2	MB2	128 bits per MB	Base + 0x00A0
Message Buffer 3	MB3	128 bits per MB	Base + 0x00B0
Message Buffer 4	MB4	128 bits per MB	Base + 0x00C0
Message Buffer 5	MB5	128 bits per MB	Base + 0x00D0
Message Buffer 6	MB6	128 bits per MB	Base + 0x00E0
Message Buffer 7	MB7	128 bits per MB	Base + 0x00F0
Message Buffer 8	MB8	128 bits per MB	Base + 0x0100
Message Buffer 9	MB9	128 bits per MB	Base + 0x0110

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 10	MB10	128 bits per MB	Base + 0x0120
Message Buffer 11	MB11	128 bits per MB	Base + 0x0130
Message Buffer 12	MB12	128 bits per MB	Base + 0x0140
Message Buffer 13	MB13	128 bits per MB	Base + 0x0150
Message Buffer 14	MB14	128 bits per MB	Base + 0x0160
Message Buffer 15	MB15	128 bits per MB	Base + 0x0170
Message Buffer 16	MB16	128 bits per MB	Base + 0x0180
Message Buffer 17	MB17	128 bits per MB	Base + 0x0190
Message Buffer 18	MB18	128 bits per MB	Base + 0x01A0
Message Buffer 19	MB19	128 bits per MB	Base + 0x01B0
Message Buffer 20	MB20	128 bits per MB	Base + 0x01C0
Message Buffer 21	MB21	128 bits per MB	Base + 0x01D0
Message Buffer 22	MB22	128 bits per MB	Base + 0x01E0
Message Buffer 23	MB23	128 bits per MB	Base + 0x01F0
Message Buffer 24	MB24	128 bits per MB	Base + 0x0200
Message Buffer 25	MB25	128 bits per MB	Base + 0x0210
Message Buffer 26	MB26	128 bits per MB	Base + 0x0220
Message Buffer 27	MB27	128 bits per MB	Base + 0x0230
Message Buffer 28	MB28	128 bits per MB	Base + 0x0240
Message Buffer 29	MB29	128 bits per MB	Base + 0x0250

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 30	MB30	128 bits per MB	Base + 0x0260
Message Buffer 31	MB31	128 bits per MB	Base + 0x0270
Message Buffer 32	MB32	128 bits per MB	Base + 0x0280
Message Buffer 33	MB33	128 bits per MB	Base + 0x0290
Message Buffer 34	MB34	128 bits per MB	Base + 0x02A0
Message Buffer 35	MB35	128 bits per MB	Base + 0x02B0
Message Buffer 36	MB36	128 bits per MB	Base + 0x02C0
Message Buffer 37	MB37	128 bits per MB	Base + 0x02D0
Message Buffer 38	MB38	128 bits per MB	Base + 0x02E0
Message Buffer 39	MB39	128 bits per MB	Base + 0x02F0
Message Buffer 40	MB40	128 bits per MB	Base + 0x0300
Message Buffer 41	MB41	128 bits per MB	Base + 0x0310
Message Buffer 42	MB42	128 bits per MB	Base + 0x0320
Message Buffer 43	MB43	128 bits per MB	Base + 0x0330
Message Buffer 44	MB44	128 bits per MB	Base + 0x0340
Message Buffer 45	MB45	128 bits per MB	Base + 0x0350
Message Buffer 46	MB46	128 bits per MB	Base + 0x0360
Message Buffer 47	MB47	128 bits per MB	Base + 0x0370
Message Buffer 48	MB48	128 bits per MB	Base + 0x0380
Message Buffer 49	MB49	128 bits per MB	Base + 0x0390

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 50	MB50	128 bits per MB	Base + 0x03A0
Message Buffer 51	MB51	128 bits per MB	Base + 0x03B0
Message Buffer 52	MB52	128 bits per MB	Base + 0x03C0
Message Buffer 53	MB53	128 bits per MB	Base + 0x03D0
Message Buffer 54	MB54	128 bits per MB	Base + 0x03E0
Message Buffer 55	MB55	128 bits per MB	Base + 0x03F0
Message Buffer 56	MB56	128 bits per MB	Base + 0x0400
Message Buffer 57	MB57	128 bits per MB	Base + 0x0410
Message Buffer 58	MB58	128 bits per MB	Base + 0x0420
Message Buffer 59	MB59	128 bits per MB	Base + 0x0430
Message Buffer 60	MB60	128 bits per MB	Base + 0x0440
Message Buffer 61	MB61	128 bits per MB	Base + 0x0450
Message Buffer 62	MB62	128 bits per MB	Base + 0x0460
Message Buffer 63	MB63	128 bits per MB	Base + 0x0470
Reserved	—	—	(Base + 0x0480) – (Base + 0x087F)
RX Individual Mask Register 0	RXIMR0	32-bit	Base + 0x0880
RX Individual Mask Register 1	RXIMR1	32-bit	Base + 0x0884
RX Individual Mask Register 2	RXIMR2	32-bit	Base + 0x0888
RX Individual Mask Register 3	RXIMR3	32-bit	Base + 0x088C
RX Individual Mask Register 4	RXIMR4	32-bit	Base + 0x0890
RX Individual Mask Register 5	RXIMR5	32-bit	Base + 0x0894
RX Individual Mask Register 6	RXIMR6	32-bit	Base + 0x0898
RX Individual Mask Register 7	RXIMR7	32-bit	Base + 0x089C
RX Individual Mask Register 8	RXIMR8	32-bit	Base + 0x08A0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 9	RXIMR9	32-bit	Base + 0x08A4
RX Individual Mask Register 10	RXIMR10	32-bit	Base + 0x08A8
RX Individual Mask Register 11	RXIMR11	32-bit	Base + 0x08AC
RX Individual Mask Register 12	RXIMR12	32-bit	Base + 0x08B0
RX Individual Mask Register 13	RXIMR13	32-bit	Base + 0x08B4
RX Individual Mask Register 14	RXIMR14	32-bit	Base + 0x08B8
RX Individual Mask Register 15	RXIMR15	32-bit	Base + 0x08BC
RX Individual Mask Register 16	RXIMR16	32-bit	Base + 0x08C0
RX Individual Mask Register 17	RXIMR17	32-bit	Base + 0x08C4
RX Individual Mask Register 18	RXIMR18	32-bit	Base + 0x08C8
RX Individual Mask Register 19	RXIMR19	32-bit	Base + 0x08CC
RX Individual Mask Register 20	RXIMR20	32-bit	Base + 0x08D0
RX Individual Mask Register 21	RXIMR21	32-bit	Base + 0x08D4
RX Individual Mask Register 22	RXIMR22	32-bit	Base + 0x08D8
RX Individual Mask Register 23	RXIMR23	32-bit	Base + 0x08DC
RX Individual Mask Register 24	RXIMR24	32-bit	Base + 0x08E0
RX Individual Mask Register 25	RXIMR25	32-bit	Base + 0x08E4
RX Individual Mask Register 26	RXIMR26	32-bit	Base + 0x08E8
RX Individual Mask Register 27	RXIMR27	32-bit	Base + 0x08EC
RX Individual Mask Register 28	RXIMR28	32-bit	Base + 0x08F0
RX Individual Mask Register 29	RXIMR29	32-bit	Base + 0x08F4
RX Individual Mask Register 30	RXIMR30	32-bit	Base + 0x08F8
RX Individual Mask Register 31	RXIMR31	32-bit	Base + 0x08FC
RX Individual Mask Register 32	RXIMR32	32-bit	Base + 0x0900
RX Individual Mask Register 33	RXIMR33	32-bit	Base + 0x0904
RX Individual Mask Register 34	RXIMR34	32-bit	Base + 0x0908
RX Individual Mask Register 35	RXIMR35	32-bit	Base + 0x090C
RX Individual Mask Register 36	RXIMR36	32-bit	Base + 0x0910
RX Individual Mask Register 37	RXIMR37	32-bit	Base + 0x0914
RX Individual Mask Register 38	RXIMR38	32-bit	Base + 0x0918
RX Individual Mask Register 39	RXIMR39	32-bit	Base + 0x091C
RX Individual Mask Register 40	RXIMR40	32-bit	Base + 0x0920
RX Individual Mask Register 41	RXIMR41	32-bit	Base + 0x0924
RX Individual Mask Register 42	RXIMR42	32-bit	Base + 0x0928

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 43	RXIMR43	32-bit	Base + 0x092C
RX Individual Mask Register 44	RXIMR44	32-bit	Base + 0x0930
RX Individual Mask Register 45	RXIMR45	32-bit	Base + 0x0934
RX Individual Mask Register 46	RXIMR46	32-bit	Base + 0x0938
RX Individual Mask Register 47	RXIMR47	32-bit	Base + 0x093C
RX Individual Mask Register 48	RXIMR48	32-bit	Base + 0x0940
RX Individual Mask Register 49	RXIMR49	32-bit	Base + 0x0944
RX Individual Mask Register 50	RXIMR50	32-bit	Base + 0x0948
RX Individual Mask Register 51	RXIMR51	32-bit	Base + 0x094C
RX Individual Mask Register 52	RXIMR52	32-bit	Base + 0x0950
RX Individual Mask Register 53	RXIMR53	32-bit	Base + 0x0954
RX Individual Mask Register 54	RXIMR54	32-bit	Base + 0x0958
RX Individual Mask Register 55	RXIMR55	32-bit	Base + 0x095C
RX Individual Mask Register 56	RXIMR56	32-bit	Base + 0x0960
RX Individual Mask Register 57	RXIMR57	32-bit	Base + 0x0964
RX Individual Mask Register 58	RXIMR58	32-bit	Base + 0x0968
RX Individual Mask Register 59	RXIMR59	32-bit	Base + 0x096C
RX Individual Mask Register 60	RXIMR60	32-bit	Base + 0x0970
RX Individual Mask Register 61	RXIMR61	32-bit	Base + 0x0974
RX Individual Mask Register 62	RXIMR62	32-bit	Base + 0x0978
RX Individual Mask Register 63	RXIMR63	32-bit	Base + 0x097C
Reserved	—	—	(Base + 0x0980) – (Base + 0x3FFF)
FlexCAN_2		**0xFFFC_8000**	
Module Configuration	MCR	32-bit	Base + 0x0000
Control Register	CTRL	32-bit	Base + 0x0004
Free Running Timer	TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x000F)
Rx Global Mask Register	RXGMASK	32-bit	Base + 0x0010
Rx 14 Mask Register	RX14MASK	32-bit	Base + 0x0014
Rx 15 Mask Register	RX15MASK	32-bit	Base + 0x0018
Error Counter Register	ECR	32-bit	Base + 0x001C
Error and Status Register	ESR	32-bit	Base + 0x0020
Interrupt Masks 2 Register	IMASK2	32-bit	Base + 0x0024

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Interrupt Masks 1 Register	IMASK1	32-bit	Base + 0x0028
Interrupt Flags 2 Register	IFLAG2	32-bit	Base + 0x002C
Interrupt Flags 1 Register	IFLAG1	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034 – 0x007F)
Message Buffer 0	MB0	128 bits per MB	Base + 0x0080
Message Buffer 1	MB1	128 bits per MB	Base + 0x0090
Message Buffer 2	MB2	128 bits per MB	Base + 0x00A0
Message Buffer 3	MB3	128 bits per MB	Base + 0x00B0
Message Buffer 4	MB4	128 bits per MB	Base + 0x00C0
Message Buffer 5	MB5	128 bits per MB	Base + 0x00D0
Message Buffer 6	MB6	128 bits per MB	Base + 0x00E0
Message Buffer 7	MB7	128 bits per MB	Base + 0x00F0
Message Buffer 8	MB8	128 bits per MB	Base + 0x0100
Message Buffer 9	MB9	128 bits per MB	Base + 0x0110
Message Buffer 10	MB10	128 bits per MB	Base + 0x0120
Message Buffer 11	MB11	128 bits per MB	Base + 0x0130
Message Buffer 12	MB12	128 bits per MB	Base + 0x0140
Message Buffer 13	MB13	128 bits per MB	Base + 0x0150
Message Buffer 14	MB14	128 bits per MB	Base + 0x0160
Message Buffer 15	MB15	128 bits per MB	Base + 0x0170
Message Buffer 16	MB16	128 bits per MB	Base + 0x0180
Message Buffer 17	MB17	128 bits per MB	Base + 0x0190

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 18	MB18	128 bits per MB	Base + 0x01A0
Message Buffer 19	MB19	128 bits per MB	Base + 0x01B0
Message Buffer 20	MB20	128 bits per MB	Base + 0x01C0
Message Buffer 21	MB21	128 bits per MB	Base + 0x01D0
Message Buffer 22	MB22	128 bits per MB	Base + 0x01E0
Message Buffer 23	MB23	128 bits per MB	Base + 0x01F0
Message Buffer 24	MB24	128 bits per MB	Base + 0x0200
Message Buffer 25	MB25	128 bits per MB	Base + 0x0210
Message Buffer 26	MB26	128 bits per MB	Base + 0x0220
Message Buffer 27	MB27	128 bits per MB	Base + 0x0230
Message Buffer 28	MB28	128 bits per MB	Base + 0x0240
Message Buffer 29	MB29	128 bits per MB	Base + 0x0250
Message Buffer 30	MB30	128 bits per MB	Base + 0x0260
Message Buffer 31	MB31	128 bits per MB	Base + 0x0270
Message Buffer 32	MB32	128 bits per MB	Base + 0x0280
Message Buffer 33	MB33	128 bits per MB	Base + 0x0290
Message Buffer 34	MB34	128 bits per MB	Base + 0x02A0
Message Buffer 35	MB35	128 bits per MB	Base + 0x02B0
Message Buffer 36	MB36	128 bits per MB	Base + 0x02C0
Message Buffer 37	MB37	128 bits per MB	Base + 0x02D0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 38	MB38	128 bits per MB	Base + 0x02E0
Message Buffer 39	MB39	128 bits per MB	Base + 0x02F0
Message Buffer 40	MB40	128 bits per MB	Base + 0x0300
Message Buffer 41	MB41	128 bits per MB	Base + 0x0310
Message Buffer 42	MB42	128 bits per MB	Base + 0x0320
Message Buffer 43	MB43	128 bits per MB	Base + 0x0330
Message Buffer 44	MB44	128 bits per MB	Base + 0x0340
Message Buffer 45	MB45	128 bits per MB	Base + 0x0350
Message Buffer 46	MB46	128 bits per MB	Base + 0x0360
Message Buffer 47	MB47	128 bits per MB	Base + 0x0370
Message Buffer 48	MB48	128 bits per MB	Base + 0x0380
Message Buffer 49	MB49	128 bits per MB	Base + 0x0390
Message Buffer 50	MB50	128 bits per MB	Base + 0x03A0
Message Buffer 51	MB51	128 bits per MB	Base + 0x03B0
Message Buffer 52	MB52	128 bits per MB	Base + 0x03C0
Message Buffer 53	MB53	128 bits per MB	Base + 0x03D0
Message Buffer 54	MB54	128 bits per MB	Base + 0x03E0
Message Buffer 55	MB55	128 bits per MB	Base + 0x03F0
Message Buffer 56	MB56	128 bits per MB	Base + 0x0400
Message Buffer 57	MB57	128 bits per MB	Base + 0x0410

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 58	MB58	128 bits per MB	Base + 0x0420
Message Buffer 59	MB59	128 bits per MB	Base + 0x0430
Message Buffer 60	MB60	128 bits per MB	Base + 0x0440
Message Buffer 61	MB61	128 bits per MB	Base + 0x0450
Message Buffer 62	MB62	128 bits per MB	Base + 0x0460
Message Buffer 63	MB63	128 bits per MB	Base + 0x0470
Reserved	—	—	(Base + 0x0480) – (Base + 0x087F)
RX Individual Mask Register 0	RXIMR0	32-bit	Base + 0x0880
RX Individual Mask Register 1	RXIMR1	32-bit	Base + 0x0884
RX Individual Mask Register 2	RXIMR2	32-bit	Base + 0x0888
RX Individual Mask Register 3	RXIMR3	32-bit	Base + 0x088C
RX Individual Mask Register 4	RXIMR4	32-bit	Base + 0x0890
RX Individual Mask Register 5	RXIMR5	32-bit	Base + 0x0894
RX Individual Mask Register 6	RXIMR6	32-bit	Base + 0x0898
RX Individual Mask Register 7	RXIMR7	32-bit	Base + 0x089C
RX Individual Mask Register 8	RXIMR8	32-bit	Base + 0x08A0
RX Individual Mask Register 9	RXIMR9	32-bit	Base + 0x08A4
RX Individual Mask Register 10	RXIMR10	32-bit	Base + 0x08A8
RX Individual Mask Register 11	RXIMR11	32-bit	Base + 0x08AC
RX Individual Mask Register 12	RXIMR12	32-bit	Base + 0x08B0
RX Individual Mask Register 13	RXIMR13	32-bit	Base + 0x08B4
RX Individual Mask Register 14	RXIMR14	32-bit	Base + 0x08B8
RX Individual Mask Register 15	RXIMR15	32-bit	Base + 0x08BC
RX Individual Mask Register 16	RXIMR16	32-bit	Base + 0x08C0
RX Individual Mask Register 17	RXIMR17	32-bit	Base + 0x08C4
RX Individual Mask Register 18	RXIMR18	32-bit	Base + 0x08C8
RX Individual Mask Register 19	RXIMR19	32-bit	Base + 0x08CC
RX Individual Mask Register 20	RXIMR20	32-bit	Base + 0x08D0
RX Individual Mask Register 21	RXIMR21	32-bit	Base + 0x08D4
RX Individual Mask Register 22	RXIMR22	32-bit	Base + 0x08D8

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 23	RXIMR23	32-bit	Base + 0x08DC
RX Individual Mask Register 24	RXIMR24	32-bit	Base + 0x08E0
RX Individual Mask Register 25	RXIMR25	32-bit	Base + 0x08E4
RX Individual Mask Register 26	RXIMR26	32-bit	Base + 0x08E8
RX Individual Mask Register 27	RXIMR27	32-bit	Base + 0x08EC
RX Individual Mask Register 28	RXIMR28	32-bit	Base + 0x08F0
RX Individual Mask Register 29	RXIMR29	32-bit	Base + 0x08F4
RX Individual Mask Register 30	RXIMR30	32-bit	Base + 0x08F8
RX Individual Mask Register 31	RXIMR31	32-bit	Base + 0x08FC
RX Individual Mask Register 32	RXIMR32	32-bit	Base + 0x0900
RX Individual Mask Register 33	RXIMR33	32-bit	Base + 0x0904
RX Individual Mask Register 34	RXIMR34	32-bit	Base + 0x0908
RX Individual Mask Register 35	RXIMR35	32-bit	Base + 0x090C
RX Individual Mask Register 36	RXIMR36	32-bit	Base + 0x0910
RX Individual Mask Register 37	RXIMR37	32-bit	Base + 0x0914
RX Individual Mask Register 38	RXIMR38	32-bit	Base + 0x0918
RX Individual Mask Register 39	RXIMR39	32-bit	Base + 0x091C
RX Individual Mask Register 40	RXIMR40	32-bit	Base + 0x0920
RX Individual Mask Register 41	RXIMR41	32-bit	Base + 0x0924
RX Individual Mask Register 42	RXIMR42	32-bit	Base + 0x0928
RX Individual Mask Register 43	RXIMR43	32-bit	Base + 0x092C
RX Individual Mask Register 44	RXIMR44	32-bit	Base + 0x0930
RX Individual Mask Register 45	RXIMR45	32-bit	Base + 0x0934
RX Individual Mask Register 46	RXIMR46	32-bit	Base + 0x0938
RX Individual Mask Register 47	RXIMR47	32-bit	Base + 0x093C
RX Individual Mask Register 48	RXIMR48	32-bit	Base + 0x0940
RX Individual Mask Register 49	RXIMR49	32-bit	Base + 0x0944
RX Individual Mask Register 50	RXIMR50	32-bit	Base + 0x0948
RX Individual Mask Register 51	RXIMR51	32-bit	Base + 0x094C
RX Individual Mask Register 52	RXIMR52	32-bit	Base + 0x0950
RX Individual Mask Register 53	RXIMR53	32-bit	Base + 0x0954
RX Individual Mask Register 54	RXIMR54	32-bit	Base + 0x0958
RX Individual Mask Register 55	RXIMR55	32-bit	Base + 0x095C
RX Individual Mask Register 56	RXIMR56	32-bit	Base + 0x0960

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 57	RXIMR57	32-bit	Base + 0x0964
RX Individual Mask Register 58	RXIMR58	32-bit	Base + 0x0968
RX Individual Mask Register 59	RXIMR59	32-bit	Base + 0x096C
RX Individual Mask Register 60	RXIMR60	32-bit	Base + 0x0970
RX Individual Mask Register 61	RXIMR61	32-bit	Base + 0x0974
RX Individual Mask Register 62	RXIMR62	32-bit	Base + 0x0978
RX Individual Mask Register 63	RXIMR63	32-bit	Base + 0x097C
Reserved	—	—	(Base + 0x0980) – (Base + 0x3FFF)
FlexCAN_3		**0xFFFC_C000**	
Module Configuration	MCR	32-bit	Base + 0x0000
Control Register	CTRL	32-bit	Base + 0x0004
Free Running Timer	TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x000F)
Rx Global Mask Register	RXGMASK	32-bit	Base + 0x0010
Rx 14 Mask Register	RX14MASK	32-bit	Base + 0x0014
Rx 15 Mask Register	RX15MASK	32-bit	Base + 0x0018
Error Counter Register	ECR	32-bit	Base + 0x001C
Error and Status Register	ESR	32-bit	Base + 0x0020
Interrupt Masks 2 Register	IMASK2	32-bit	Base + 0x0024
Interrupt Masks 1 Register	IMASK1	32-bit	Base + 0x0028
Interrupt Flags 2 Register	IFLAG2	32-bit	Base + 0x002C
Interrupt Flags 1 Register	IFLAG1	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034 – 0x007F)
Message Buffer 0	MB0	128 bits per MB	Base + 0x0080
Message Buffer 1	MB1	128 bits per MB	Base + 0x0090
Message Buffer 2	MB2	128 bits per MB	Base + 0x00A0
Message Buffer 3	MB3	128 bits per MB	Base + 0x00B0
Message Buffer 4	MB4	128 bits per MB	Base + 0x00C0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 5	MB5	128 bits per MB	Base + 0x00D0
Message Buffer 6	MB6	128 bits per MB	Base + 0x00E0
Message Buffer 7	MB7	128 bits per MB	Base + 0x00F0
Message Buffer 8	MB8	128 bits per MB	Base + 0x0100
Message Buffer 9	MB9	128 bits per MB	Base + 0x0110
Message Buffer 10	MB10	128 bits per MB	Base + 0x0120
Message Buffer 11	MB11	128 bits per MB	Base + 0x0130
Message Buffer 12	MB12	128 bits per MB	Base + 0x0140
Message Buffer 13	MB13	128 bits per MB	Base + 0x0150
Message Buffer 14	MB14	128 bits per MB	Base + 0x0160
Message Buffer 15	MB15	128 bits per MB	Base + 0x0170
Message Buffer 16	MB16	128 bits per MB	Base + 0x0180
Message Buffer 17	MB17	128 bits per MB	Base + 0x0190
Message Buffer 18	MB18	128 bits per MB	Base + 0x01A0
Message Buffer 19	MB19	128 bits per MB	Base + 0x01B0
Message Buffer 20	MB20	128 bits per MB	Base + 0x01C0
Message Buffer 21	MB21	128 bits per MB	Base + 0x01D0
Message Buffer 22	MB22	128 bits per MB	Base + 0x01E0
Message Buffer 23	MB23	128 bits per MB	Base + 0x01F0
Message Buffer 24	MB24	128 bits per MB	Base + 0x0200

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 25	MB25	128 bits per MB	Base + 0x0210
Message Buffer 26	MB26	128 bits per MB	Base + 0x0220
Message Buffer 27	MB27	128 bits per MB	Base + 0x0230
Message Buffer 28	MB28	128 bits per MB	Base + 0x0240
Message Buffer 29	MB29	128 bits per MB	Base + 0x0250
Message Buffer 30	MB30	128 bits per MB	Base + 0x0260
Message Buffer 31	MB31	128 bits per MB	Base + 0x0270
Message Buffer 32	MB32	128 bits per MB	Base + 0x0280
Message Buffer 33	MB33	128 bits per MB	Base + 0x0290
Message Buffer 34	MB34	128 bits per MB	Base + 0x02A0
Message Buffer 35	MB35	128 bits per MB	Base + 0x02B0
Message Buffer 36	MB36	128 bits per MB	Base + 0x02C0
Message Buffer 37	MB37	128 bits per MB	Base + 0x02D0
Message Buffer 38	MB38	128 bits per MB	Base + 0x02E0
Message Buffer 39	MB39	128 bits per MB	Base + 0x02F0
Message Buffer 40	MB40	128 bits per MB	Base + 0x0300
Message Buffer 41	MB41	128 bits per MB	Base + 0x0310
Message Buffer 42	MB42	128 bits per MB	Base + 0x0320
Message Buffer 43	MB43	128 bits per MB	Base + 0x0330
Message Buffer 44	MB44	128 bits per MB	Base + 0x0340

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 45	MB45	128 bits per MB	Base + 0x0350
Message Buffer 46	MB46	128 bits per MB	Base + 0x0360
Message Buffer 47	MB47	128 bits per MB	Base + 0x0370
Message Buffer 48	MB48	128 bits per MB	Base + 0x0380
Message Buffer 49	MB49	128 bits per MB	Base + 0x0390
Message Buffer 50	MB50	128 bits per MB	Base + 0x03A0
Message Buffer 51	MB51	128 bits per MB	Base + 0x03B0
Message Buffer 52	MB52	128 bits per MB	Base + 0x03C0
Message Buffer 53	MB53	128 bits per MB	Base + 0x03D0
Message Buffer 54	MB54	128 bits per MB	Base + 0x03E0
Message Buffer 55	MB55	128 bits per MB	Base + 0x03F0
Message Buffer 56	MB56	128 bits per MB	Base + 0x0400
Message Buffer 57	MB57	128 bits per MB	Base + 0x0410
Message Buffer 58	MB58	128 bits per MB	Base + 0x0420
Message Buffer 59	MB59	128 bits per MB	Base + 0x0430
Message Buffer 60	MB60	128 bits per MB	Base + 0x0440
Message Buffer 61	MB61	128 bits per MB	Base + 0x0450
Message Buffer 62	MB62	128 bits per MB	Base + 0x0460
Message Buffer 63	MB63	128 bits per MB	Base + 0x0470
Reserved	—	—	(Base + 0x0480) – (Base + 0x087F)
RX Individual Mask Register 0	RXIMR0	32-bit	Base + 0x0880

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 1	RXIMR1	32-bit	Base + 0x0884
RX Individual Mask Register 2	RXIMR2	32-bit	Base + 0x0888
RX Individual Mask Register 3	RXIMR3	32-bit	Base + 0x088C
RX Individual Mask Register 4	RXIMR4	32-bit	Base + 0x0890
RX Individual Mask Register 5	RXIMR5	32-bit	Base + 0x0894
RX Individual Mask Register 6	RXIMR6	32-bit	Base + 0x0898
RX Individual Mask Register 7	RXIMR7	32-bit	Base + 0x089C
RX Individual Mask Register 8	RXIMR8	32-bit	Base + 0x08A0
RX Individual Mask Register 9	RXIMR9	32-bit	Base + 0x08A4
RX Individual Mask Register 10	RXIMR10	32-bit	Base + 0x08A8
RX Individual Mask Register 11	RXIMR11	32-bit	Base + 0x08AC
RX Individual Mask Register 12	RXIMR12	32-bit	Base + 0x08B0
RX Individual Mask Register 13	RXIMR13	32-bit	Base + 0x08B4
RX Individual Mask Register 14	RXIMR14	32-bit	Base + 0x08B8
RX Individual Mask Register 15	RXIMR15	32-bit	Base + 0x08BC
RX Individual Mask Register 16	RXIMR16	32-bit	Base + 0x08C0
RX Individual Mask Register 17	RXIMR17	32-bit	Base + 0x08C4
RX Individual Mask Register 18	RXIMR18	32-bit	Base + 0x08C8
RX Individual Mask Register 19	RXIMR19	32-bit	Base + 0x08CC
RX Individual Mask Register 20	RXIMR20	32-bit	Base + 0x08D0
RX Individual Mask Register 21	RXIMR21	32-bit	Base + 0x08D4
RX Individual Mask Register 22	RXIMR22	32-bit	Base + 0x08D8
RX Individual Mask Register 23	RXIMR23	32-bit	Base + 0x08DC
RX Individual Mask Register 24	RXIMR24	32-bit	Base + 0x08E0
RX Individual Mask Register 25	RXIMR25	32-bit	Base + 0x08E4
RX Individual Mask Register 26	RXIMR26	32-bit	Base + 0x08E8
RX Individual Mask Register 27	RXIMR27	32-bit	Base + 0x08EC
RX Individual Mask Register 28	RXIMR28	32-bit	Base + 0x08F0
RX Individual Mask Register 29	RXIMR29	32-bit	Base + 0x08F4
RX Individual Mask Register 30	RXIMR30	32-bit	Base + 0x08F8
RX Individual Mask Register 31	RXIMR31	32-bit	Base + 0x08FC
RX Individual Mask Register 32	RXIMR32	32-bit	Base + 0x0900
RX Individual Mask Register 33	RXIMR33	32-bit	Base + 0x0904
RX Individual Mask Register 34	RXIMR34	32-bit	Base + 0x0908

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 35	RXIMR35	32-bit	Base + 0x090C
RX Individual Mask Register 36	RXIMR36	32-bit	Base + 0x0910
RX Individual Mask Register 37	RXIMR37	32-bit	Base + 0x0914
RX Individual Mask Register 38	RXIMR38	32-bit	Base + 0x0918
RX Individual Mask Register 39	RXIMR39	32-bit	Base + 0x091C
RX Individual Mask Register 40	RXIMR40	32-bit	Base + 0x0920
RX Individual Mask Register 41	RXIMR41	32-bit	Base + 0x0924
RX Individual Mask Register 42	RXIMR42	32-bit	Base + 0x0928
RX Individual Mask Register 43	RXIMR43	32-bit	Base + 0x092C
RX Individual Mask Register 44	RXIMR44	32-bit	Base + 0x0930
RX Individual Mask Register 45	RXIMR45	32-bit	Base + 0x0934
RX Individual Mask Register 46	RXIMR46	32-bit	Base + 0x0938
RX Individual Mask Register 47	RXIMR47	32-bit	Base + 0x093C
RX Individual Mask Register 48	RXIMR48	32-bit	Base + 0x0940
RX Individual Mask Register 49	RXIMR49	32-bit	Base + 0x0944
RX Individual Mask Register 50	RXIMR50	32-bit	Base + 0x0948
RX Individual Mask Register 51	RXIMR51	32-bit	Base + 0x094C
RX Individual Mask Register 52	RXIMR52	32-bit	Base + 0x0950
RX Individual Mask Register 53	RXIMR53	32-bit	Base + 0x0954
RX Individual Mask Register 54	RXIMR54	32-bit	Base + 0x0958
RX Individual Mask Register 55	RXIMR55	32-bit	Base + 0x095C
RX Individual Mask Register 56	RXIMR56	32-bit	Base + 0x0960
RX Individual Mask Register 57	RXIMR57	32-bit	Base + 0x0964
RX Individual Mask Register 58	RXIMR58	32-bit	Base + 0x0968
RX Individual Mask Register 59	RXIMR59	32-bit	Base + 0x096C
RX Individual Mask Register 60	RXIMR60	32-bit	Base + 0x0970
RX Individual Mask Register 61	RXIMR61	32-bit	Base + 0x0974
RX Individual Mask Register 62	RXIMR62	32-bit	Base + 0x0978
RX Individual Mask Register 63	RXIMR63	32-bit	Base + 0x097C
Reserved	—	—	(Base + 0x0980) – (Base + 0x3FFF)
FlexCAN_4		**0xFFFD_0000**	
Module Configuration	MCR	32-bit	Base + 0x0000
Control Register	CTRL	32-bit	Base + 0x0004

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Free Running Timer	TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x000F)
Rx Global Mask Register	RXGMASK	32-bit	Base + 0x0010
Rx 14 Mask Register	RX14MASK	32-bit	Base + 0x0014
Rx 15 Mask Register	RX15MASK	32-bit	Base + 0x0018
Error Counter Register	ECR	32-bit	Base + 0x001C
Error and Status Register	ESR	32-bit	Base + 0x0020
Interrupt Masks 2 Register	IMASK2	32-bit	Base + 0x0024
Interrupt Masks 1 Register	IMASK1	32-bit	Base + 0x0028
Interrupt Flags 2 Register	IFLAG2	32-bit	Base + 0x002C
Interrupt Flags 1 Register	IFLAG1	32-bit	Base + 0x0030
Reserved	—	—	Base + (0x0034 – 0x007F)
Message Buffer 0	MB0	128 bits per MB	Base + 0x0080
Message Buffer 1	MB1	128 bits per MB	Base + 0x0090
Message Buffer 2	MB2	128 bits per MB	Base + 0x00A0
Message Buffer 3	MB3	128 bits per MB	Base + 0x00B0
Message Buffer 4	MB4	128 bits per MB	Base + 0x00C0
Message Buffer 5	MB5	128 bits per MB	Base + 0x00D0
Message Buffer 6	MB6	128 bits per MB	Base + 0x00E0
Message Buffer 7	MB7	128 bits per MB	Base + 0x00F0
Message Buffer 8	MB8	128 bits per MB	Base + 0x0100
Message Buffer 9	MB9	128 bits per MB	Base + 0x0110
Message Buffer 10	MB10	128 bits per MB	Base + 0x0120
Message Buffer 11	MB11	128 bits per MB	Base + 0x0130

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 12	MB12	128 bits per MB	Base + 0x0140
Message Buffer 13	MB13	128 bits per MB	Base + 0x0150
Message Buffer 14	MB14	128 bits per MB	Base + 0x0160
Message Buffer 15	MB15	128 bits per MB	Base + 0x0170
Message Buffer 16	MB16	128 bits per MB	Base + 0x0180
Message Buffer 17	MB17	128 bits per MB	Base + 0x0190
Message Buffer 18	MB18	128 bits per MB	Base + 0x01A0
Message Buffer 19	MB19	128 bits per MB	Base + 0x01B0
Message Buffer 20	MB20	128 bits per MB	Base + 0x01C0
Message Buffer 21	MB21	128 bits per MB	Base + 0x01D0
Message Buffer 22	MB22	128 bits per MB	Base + 0x01E0
Message Buffer 23	MB23	128 bits per MB	Base + 0x01F0
Message Buffer 24	MB24	128 bits per MB	Base + 0x0200
Message Buffer 25	MB25	128 bits per MB	Base + 0x0210
Message Buffer 26	MB26	128 bits per MB	Base + 0x0220
Message Buffer 27	MB27	128 bits per MB	Base + 0x0230
Message Buffer 28	MB28	128 bits per MB	Base + 0x0240
Message Buffer 29	MB29	128 bits per MB	Base + 0x0250
Message Buffer 30	MB30	128 bits per MB	Base + 0x0260
Message Buffer 31	MB31	128 bits per MB	Base + 0x0270

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 32	MB32	128 bits per MB	Base + 0x0280
Message Buffer 33	MB33	128 bits per MB	Base + 0x0290
Message Buffer 34	MB34	128 bits per MB	Base + 0x02A0
Message Buffer 35	MB35	128 bits per MB	Base + 0x02B0
Message Buffer 36	MB36	128 bits per MB	Base + 0x02C0
Message Buffer 37	MB37	128 bits per MB	Base + 0x02D0
Message Buffer 38	MB38	128 bits per MB	Base + 0x02E0
Message Buffer 39	MB39	128 bits per MB	Base + 0x02F0
Message Buffer 40	MB40	128 bits per MB	Base + 0x0300
Message Buffer 41	MB41	128 bits per MB	Base + 0x0310
Message Buffer 42	MB42	128 bits per MB	Base + 0x0320
Message Buffer 43	MB43	128 bits per MB	Base + 0x0330
Message Buffer 44	MB44	128 bits per MB	Base + 0x0340
Message Buffer 45	MB45	128 bits per MB	Base + 0x0350
Message Buffer 46	MB46	128 bits per MB	Base + 0x0360
Message Buffer 47	MB47	128 bits per MB	Base + 0x0370
Message Buffer 48	MB48	128 bits per MB	Base + 0x0380
Message Buffer 49	MB49	128 bits per MB	Base + 0x0390
Message Buffer 50	MB50	128 bits per MB	Base + 0x03A0
Message Buffer 51	MB51	128 bits per MB	Base + 0x03B0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 52	MB52	128 bits per MB	Base + 0x03C0
Message Buffer 53	MB53	128 bits per MB	Base + 0x03D0
Message Buffer 54	MB54	128 bits per MB	Base + 0x03E0
Message Buffer 55	MB55	128 bits per MB	Base + 0x03F0
Message Buffer 56	MB56	128 bits per MB	Base + 0x0400
Message Buffer 57	MB57	128 bits per MB	Base + 0x0410
Message Buffer 58	MB58	128 bits per MB	Base + 0x0420
Message Buffer 59	MB59	128 bits per MB	Base + 0x0430
Message Buffer 60	MB60	128 bits per MB	Base + 0x0440
Message Buffer 61	MB61	128 bits per MB	Base + 0x0450
Message Buffer 62	MB62	128 bits per MB	Base + 0x0460
Message Buffer 63	MB63	128 bits per MB	Base + 0x0470
Reserved	—	—	(Base + 0x0480) – (Base + 0x087F)
RX Individual Mask Register 0	RXIMR0	32-bit	Base + 0x0880
RX Individual Mask Register 1	RXIMR1	32-bit	Base + 0x0884
RX Individual Mask Register 2	RXIMR2	32-bit	Base + 0x0888
RX Individual Mask Register 3	RXIMR3	32-bit	Base + 0x088C
RX Individual Mask Register 4	RXIMR4	32-bit	Base + 0x0890
RX Individual Mask Register 5	RXIMR5	32-bit	Base + 0x0894
RX Individual Mask Register 6	RXIMR6	32-bit	Base + 0x0898
RX Individual Mask Register 7	RXIMR7	32-bit	Base + 0x089C
RX Individual Mask Register 8	RXIMR8	32-bit	Base + 0x08A0
RX Individual Mask Register 9	RXIMR9	32-bit	Base + 0x08A4
RX Individual Mask Register 10	RXIMR10	32-bit	Base + 0x08A8
RX Individual Mask Register 11	RXIMR11	32-bit	Base + 0x08AC
RX Individual Mask Register 12	RXIMR12	32-bit	Base + 0x08B0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 13	RXIMR13	32-bit	Base + 0x08B4
RX Individual Mask Register 14	RXIMR14	32-bit	Base + 0x08B8
RX Individual Mask Register 15	RXIMR15	32-bit	Base + 0x08BC
RX Individual Mask Register 16	RXIMR16	32-bit	Base + 0x08C0
RX Individual Mask Register 17	RXIMR17	32-bit	Base + 0x08C4
RX Individual Mask Register 18	RXIMR18	32-bit	Base + 0x08C8
RX Individual Mask Register 19	RXIMR19	32-bit	Base + 0x08CC
RX Individual Mask Register 20	RXIMR20	32-bit	Base + 0x08D0
RX Individual Mask Register 21	RXIMR21	32-bit	Base + 0x08D4
RX Individual Mask Register 22	RXIMR22	32-bit	Base + 0x08D8
RX Individual Mask Register 23	RXIMR23	32-bit	Base + 0x08DC
RX Individual Mask Register 24	RXIMR24	32-bit	Base + 0x08E0
RX Individual Mask Register 25	RXIMR25	32-bit	Base + 0x08E4
RX Individual Mask Register 26	RXIMR26	32-bit	Base + 0x08E8
RX Individual Mask Register 27	RXIMR27	32-bit	Base + 0x08EC
RX Individual Mask Register 28	RXIMR28	32-bit	Base + 0x08F0
RX Individual Mask Register 29	RXIMR29	32-bit	Base + 0x08F4
RX Individual Mask Register 30	RXIMR30	32-bit	Base + 0x08F8
RX Individual Mask Register 31	RXIMR31	32-bit	Base + 0x08FC
RX Individual Mask Register 32	RXIMR32	32-bit	Base + 0x0900
RX Individual Mask Register 33	RXIMR33	32-bit	Base + 0x0904
RX Individual Mask Register 34	RXIMR34	32-bit	Base + 0x0908
RX Individual Mask Register 35	RXIMR35	32-bit	Base + 0x090C
RX Individual Mask Register 36	RXIMR36	32-bit	Base + 0x0910
RX Individual Mask Register 37	RXIMR37	32-bit	Base + 0x0914
RX Individual Mask Register 38	RXIMR38	32-bit	Base + 0x0918
RX Individual Mask Register 39	RXIMR39	32-bit	Base + 0x091C
RX Individual Mask Register 40	RXIMR40	32-bit	Base + 0x0920
RX Individual Mask Register 41	RXIMR41	32-bit	Base + 0x0924
RX Individual Mask Register 42	RXIMR42	32-bit	Base + 0x0928
RX Individual Mask Register 43	RXIMR43	32-bit	Base + 0x092C
RX Individual Mask Register 44	RXIMR44	32-bit	Base + 0x0930
RX Individual Mask Register 45	RXIMR45	32-bit	Base + 0x0934
RX Individual Mask Register 46	RXIMR46	32-bit	Base + 0x0938

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 47	RXIMR47	32-bit	Base + 0x093C
RX Individual Mask Register 48	RXIMR48	32-bit	Base + 0x0940
RX Individual Mask Register 49	RXIMR49	32-bit	Base + 0x0944
RX Individual Mask Register 50	RXIMR50	32-bit	Base + 0x0948
RX Individual Mask Register 51	RXIMR51	32-bit	Base + 0x094C
RX Individual Mask Register 52	RXIMR52	32-bit	Base + 0x0950
RX Individual Mask Register 53	RXIMR53	32-bit	Base + 0x0954
RX Individual Mask Register 54	RXIMR54	32-bit	Base + 0x0958
RX Individual Mask Register 55	RXIMR55	32-bit	Base + 0x095C
RX Individual Mask Register 56	RXIMR56	32-bit	Base + 0x0960
RX Individual Mask Register 57	RXIMR57	32-bit	Base + 0x0964
RX Individual Mask Register 58	RXIMR58	32-bit	Base + 0x0968
RX Individual Mask Register 59	RXIMR59	32-bit	Base + 0x096C
RX Individual Mask Register 60	RXIMR60	32-bit	Base + 0x0970
RX Individual Mask Register 61	RXIMR61	32-bit	Base + 0x0974
RX Individual Mask Register 62	RXIMR62	32-bit	Base + 0x0978
RX Individual Mask Register 63	RXIMR63	32-bit	Base + 0x097C
Reserved	—	—	(Base + 0x0980) – (Base + 0x3FFF)
FlexCAN_5		**0xFFFD_4000**	
Module Configuration	MCR	32-bit	Base + 0x0000
Control Register	CTRL	32-bit	Base + 0x0004
Free Running Timer	TIMER	32-bit	Base + 0x0008
Reserved	—	—	Base + (0x000C – 0x000F)
Rx Global Mask Register	RXGMASK	32-bit	Base + 0x0010
Rx 14 Mask Register	RX14MASK	32-bit	Base + 0x0014
Rx 15 Mask Register	RX15MASK	32-bit	Base + 0x0018
Error Counter Register	ECR	32-bit	Base + 0x001C
Error and Status Register	ESR	32-bit	Base + 0x0020
Interrupt Masks 2 Register	IMASK2	32-bit	Base + 0x0024
Interrupt Masks 1 Register	IMASK1	32-bit	Base + 0x0028
Interrupt Flags 2 Register	IFLAG2	32-bit	Base + 0x002C
Interrupt Flags 1 Register	IFLAG1	32-bit	Base + 0x0030

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Reserved	—	—	Base + (0x0034 – 0x007F)
Message Buffer 0	MB0	128 bits per MB	Base + 0x0080
Message Buffer 1	MB1	128 bits per MB	Base + 0x0090
Message Buffer 2	MB2	128 bits per MB	Base + 0x00A0
Message Buffer 3	MB3	128 bits per MB	Base + 0x00B0
Message Buffer 4	MB4	128 bits per MB	Base + 0x00C0
Message Buffer 5	MB5	128 bits per MB	Base + 0x00D0
Message Buffer 6	MB6	128 bits per MB	Base + 0x00E0
Message Buffer 7	MB7	128 bits per MB	Base + 0x00F0
Message Buffer 8	MB8	128 bits per MB	Base + 0x0100
Message Buffer 9	MB9	128 bits per MB	Base + 0x0110
Message Buffer 10	MB10	128 bits per MB	Base + 0x0120
Message Buffer 11	MB11	128 bits per MB	Base + 0x0130
Message Buffer 12	MB12	128 bits per MB	Base + 0x0140
Message Buffer 13	MB13	128 bits per MB	Base + 0x0150
Message Buffer 14	MB14	128 bits per MB	Base + 0x0160
Message Buffer 15	MB15	128 bits per MB	Base + 0x0170
Message Buffer 16	MB16	128 bits per MB	Base + 0x0180
Message Buffer 17	MB17	128 bits per MB	Base + 0x0190
Message Buffer 18	MB18	128 bits per MB	Base + 0x01A0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 19	MB19	128 bits per MB	Base + 0x01B0
Message Buffer 20	MB20	128 bits per MB	Base + 0x01C0
Message Buffer 21	MB21	128 bits per MB	Base + 0x01D0
Message Buffer 22	MB22	128 bits per MB	Base + 0x01E0
Message Buffer 23	MB23	128 bits per MB	Base + 0x01F0
Message Buffer 24	MB24	128 bits per MB	Base + 0x0200
Message Buffer 25	MB25	128 bits per MB	Base + 0x0210
Message Buffer 26	MB26	128 bits per MB	Base + 0x0220
Message Buffer 27	MB27	128 bits per MB	Base + 0x0230
Message Buffer 28	MB28	128 bits per MB	Base + 0x0240
Message Buffer 29	MB29	128 bits per MB	Base + 0x0250
Message Buffer 30	MB30	128 bits per MB	Base + 0x0260
Message Buffer 31	MB31	128 bits per MB	Base + 0x0270
Message Buffer 32	MB32	128 bits per MB	Base + 0x0280
Message Buffer 33	MB33	128 bits per MB	Base + 0x0290
Message Buffer 34	MB34	128 bits per MB	Base + 0x02A0
Message Buffer 35	MB35	128 bits per MB	Base + 0x02B0
Message Buffer 36	MB36	128 bits per MB	Base + 0x02C0
Message Buffer 37	MB37	128 bits per MB	Base + 0x02D0
Message Buffer 38	MB38	128 bits per MB	Base + 0x02E0

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 39	MB39	128 bits per MB	Base + 0x02F0
Message Buffer 40	MB40	128 bits per MB	Base + 0x0300
Message Buffer 41	MB41	128 bits per MB	Base + 0x0310
Message Buffer 42	MB42	128 bits per MB	Base + 0x0320
Message Buffer 43	MB43	128 bits per MB	Base + 0x0330
Message Buffer 44	MB44	128 bits per MB	Base + 0x0340
Message Buffer 45	MB45	128 bits per MB	Base + 0x0350
Message Buffer 46	MB46	128 bits per MB	Base + 0x0360
Message Buffer 47	MB47	128 bits per MB	Base + 0x0370
Message Buffer 48	MB48	128 bits per MB	Base + 0x0380
Message Buffer 49	MB49	128 bits per MB	Base + 0x0390
Message Buffer 50	MB50	128 bits per MB	Base + 0x03A0
Message Buffer 51	MB51	128 bits per MB	Base + 0x03B0
Message Buffer 52	MB52	128 bits per MB	Base + 0x03C0
Message Buffer 53	MB53	128 bits per MB	Base + 0x03D0
Message Buffer 54	MB54	128 bits per MB	Base + 0x03E0
Message Buffer 55	MB55	128 bits per MB	Base + 0x03F0
Message Buffer 56	MB56	128 bits per MB	Base + 0x0400
Message Buffer 57	MB57	128 bits per MB	Base + 0x0410
Message Buffer 58	MB58	128 bits per MB	Base + 0x0420

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
Message Buffer 59	MB59	128 bits per MB	Base + 0x0430
Message Buffer 60	MB60	128 bits per MB	Base + 0x0440
Message Buffer 61	MB61	128 bits per MB	Base + 0x0450
Message Buffer 62	MB62	128 bits per MB	Base + 0x0460
Message Buffer 63	MB63	128 bits per MB	Base + 0x0470
Reserved	—	—	(Base + 0x0480) – (Base + 0x087F)
RX Individual Mask Register 0	RXIMR0	32-bit	Base + 0x0880
RX Individual Mask Register 1	RXIMR1	32-bit	Base + 0x0884
RX Individual Mask Register 2	RXIMR2	32-bit	Base + 0x0888
RX Individual Mask Register 3	RXIMR3	32-bit	Base + 0x088C
RX Individual Mask Register 4	RXIMR4	32-bit	Base + 0x0890
RX Individual Mask Register 5	RXIMR5	32-bit	Base + 0x0894
RX Individual Mask Register 6	RXIMR6	32-bit	Base + 0x0898
RX Individual Mask Register 7	RXIMR7	32-bit	Base + 0x089C
RX Individual Mask Register 8	RXIMR8	32-bit	Base + 0x08A0
RX Individual Mask Register 9	RXIMR9	32-bit	Base + 0x08A4
RX Individual Mask Register 10	RXIMR10	32-bit	Base + 0x08A8
RX Individual Mask Register 11	RXIMR11	32-bit	Base + 0x08AC
RX Individual Mask Register 12	RXIMR12	32-bit	Base + 0x08B0
RX Individual Mask Register 13	RXIMR13	32-bit	Base + 0x08B4
RX Individual Mask Register 14	RXIMR14	32-bit	Base + 0x08B8
RX Individual Mask Register 15	RXIMR15	32-bit	Base + 0x08BC
RX Individual Mask Register 16	RXIMR16	32-bit	Base + 0x08C0
RX Individual Mask Register 17	RXIMR17	32-bit	Base + 0x08C4
RX Individual Mask Register 18	RXIMR18	32-bit	Base + 0x08C8
RX Individual Mask Register 19	RXIMR19	32-bit	Base + 0x08CC
RX Individual Mask Register 20	RXIMR20	32-bit	Base + 0x08D0
RX Individual Mask Register 21	RXIMR21	32-bit	Base + 0x08D4
RX Individual Mask Register 22	RXIMR22	32-bit	Base + 0x08D8
RX Individual Mask Register 23	RXIMR23	32-bit	Base + 0x08DC

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 24	RXIMR24	32-bit	Base + 0x08E0
RX Individual Mask Register 25	RXIMR25	32-bit	Base + 0x08E4
RX Individual Mask Register 26	RXIMR26	32-bit	Base + 0x08E8
RX Individual Mask Register 27	RXIMR27	32-bit	Base + 0x08EC
RX Individual Mask Register 28	RXIMR28	32-bit	Base + 0x08F0
RX Individual Mask Register 29	RXIMR29	32-bit	Base + 0x08F4
RX Individual Mask Register 30	RXIMR30	32-bit	Base + 0x08F8
RX Individual Mask Register 31	RXIMR31	32-bit	Base + 0x08FC
RX Individual Mask Register 32	RXIMR32	32-bit	Base + 0x0900
RX Individual Mask Register 33	RXIMR33	32-bit	Base + 0x0904
RX Individual Mask Register 34	RXIMR34	32-bit	Base + 0x0908
RX Individual Mask Register 35	RXIMR35	32-bit	Base + 0x090C
RX Individual Mask Register 36	RXIMR36	32-bit	Base + 0x0910
RX Individual Mask Register 37	RXIMR37	32-bit	Base + 0x0914
RX Individual Mask Register 38	RXIMR38	32-bit	Base + 0x0918
RX Individual Mask Register 39	RXIMR39	32-bit	Base + 0x091C
RX Individual Mask Register 40	RXIMR40	32-bit	Base + 0x0920
RX Individual Mask Register 41	RXIMR41	32-bit	Base + 0x0924
RX Individual Mask Register 42	RXIMR42	32-bit	Base + 0x0928
RX Individual Mask Register 43	RXIMR43	32-bit	Base + 0x092C
RX Individual Mask Register 44	RXIMR44	32-bit	Base + 0x0930
RX Individual Mask Register 45	RXIMR45	32-bit	Base + 0x0934
RX Individual Mask Register 46	RXIMR46	32-bit	Base + 0x0938
RX Individual Mask Register 47	RXIMR47	32-bit	Base + 0x093C
RX Individual Mask Register 48	RXIMR48	32-bit	Base + 0x0940
RX Individual Mask Register 49	RXIMR49	32-bit	Base + 0x0944
RX Individual Mask Register 50	RXIMR50	32-bit	Base + 0x0948
RX Individual Mask Register 51	RXIMR51	32-bit	Base + 0x094C
RX Individual Mask Register 52	RXIMR52	32-bit	Base + 0x0950
RX Individual Mask Register 53	RXIMR53	32-bit	Base + 0x0954
RX Individual Mask Register 54	RXIMR54	32-bit	Base + 0x0958
RX Individual Mask Register 55	RXIMR55	32-bit	Base + 0x095C
RX Individual Mask Register 56	RXIMR56	32-bit	Base + 0x0960
RX Individual Mask Register 57	RXIMR57	32-bit	Base + 0x0964

Table 432. Detailed register map (continued)

Register description	Register name	Used size	Address
RX Individual Mask Register 58	RXIMR58	32-bit	Base + 0x0968
RX Individual Mask Register 59	RXIMR59	32-bit	Base + 0x096C
RX Individual Mask Register 60	RXIMR60	32-bit	Base + 0x0970
RX Individual Mask Register 61	RXIMR61	32-bit	Base + 0x0974
RX Individual Mask Register 62	RXIMR62	32-bit	Base + 0x0978
RX Individual Mask Register 63	RXIMR63	32-bit	Base + 0x097C
Reserved	—	—	(Base + 0x0980) – (Base + 0x3FFF)

Revision history

Table 433. Document revision history

Date	Revision	Changes
01-Apr-2008	1	Initial release
19-Feb-2009	2	Cover page: Done the same progressive numbering of Figure and Table Throgh whole document: - Harmonized the name of the 4 different clock source with the name listened here: - FXOSC or fast external crystal oscillator 4-16 MHz - FIRC or fast internal RC oscillator 16 MHz - SIRC or slow internal RC oscillator 128 KHz - SXOSC or slow external crystal oscillator 32 KHz - FMPLL or frequency modulated phase locked loop - Harmonized the cross reference to sections. - Replaced every “Miscellaneous Control Module” or “MCM“ occurrences rispectively with “Error Correction Status Module“ or “(ECSM)“. *Chapter 1 Overview* - *Section 1.1 Features*: replaced with a new one. - *Table 2*: Added rows, extended the SRAM memory to 48 KB. - *Table 4*: removed the cell of “AP“. *Chapter 2 Signal description* - *Section 2.2 Package pinouts*: updated all pin map. - *Section 2.3 Pad configuration during reset phases* and *Section 2.6 System pins*: Updated because After power-up phase the majority of pins is in tristate end not inpu waek pull-up. - *Table 6*: - PC[1] type changed from “F“ to “M“. - footnote 9: included also PH[9:10] among the exepted pins. - *Section 2.5 Pad types*: Changed the Note . *Chapter 3 Clock description*: - Removed the reference to normal end test access, all accesses are seen as supervisor. - *Table 9*: Replaced PIT_RIT with PIT. - *Table 10*: - Replaced “ENABLE“ heading rows with “ME_GS.S_XOSC“. - Replaced “BYP“ heading rows with “OSC_CTL.OSCBYOP“. - Replaced “Hiz‘ with “High Z“. - *Table 12*: - Replaced “ENABLE“ heading rows with “OSC_CTL.S_OSC“. - Replaced “BYP“ heading rows with “OSC_CTL.OSCBYOP“. - Replaced “Hiz‘ with “High Z“. - *Table 23*: Removed FLCI_A field. - *Figure 10*: Removed MODE and DIV4 path. - *Section 3.3, “Clock Generation Module (MC_CGM)”*: Replaced with a new section. *Chapter 5 Mode Entry Module (MC_ME)*: Replaced with a new chapter.

Table 433. Document revision history (continued)

Date	Revision	Changes
19-Feb-2009	2 (continued)	*Chapter 6 Boot Assist Module (BAM)*: - Aligned naming of LINFlex module. - *Section BAM resources*: Removed any references to STM, CMU and FMPLL. *Chapter 7 Reset Generation Module (MC_RGM)*: Replaced with a new chapter. *Chapter 8 System Integration Unit Lite (SIUL)*: - Repleced the number of I/O pins from “121“ to “123“ for 144-pin and 208-pin packages. - Repleced the number of I/O pins from “77“ to “79“ for 100-pin packages. - *Table 77*: modified the reset value for bit 28:31 to “0“. - *Table 83*: changed the size of the field “SRC“ form 2 to 1 bit. - *Table 86*: Changed the definition of PCRx.SRC. *Chapter 9 Power Control Unit (MC_PCU)*: Replaced with a new chapter. *Chapter 11 e200z0h Core*: Replaced all e200z0 e200z1 occurrences with e200z0h. *Chapter 15 Error Correction Status Module (ECSM)*‘: - removed MRSR register and descibed as reserved. - removed section “13.4.3 High Priority Enables“. *Table 124*: removed MRSR register and descibed as reserved. *Section 16.6 External Signal Description*: - Updated the period since all 4 JTAG pin are shared with GPIO. - *Table 145*: updated DC field description. - *Section 16.8.4 JTAGC Instructions*: Removed Cut.1 information. *Chapter 17 Nexus Development Interface (NDI)*: - Removed references to JCOMP. - Removed section “Nexus Reset Control“. *Chapter 18 Static RAM (SRAM)*: - Updated the size of the RAM from 38 to 42KB. - *Section 18.6 Initialization and application information*: Reformatted. *Chapter 19 Flash memory*: - *Table 166* Updated. - *Section 19.4.1 Introduction*: Replaced “SPP” with “RPP”. - Removed figure”FLASH Memory Controller Block Diagram”. - *Table 230*: Replaced the reset value with which ones defined in the table footnote and removed them. *Chapter 20 Deserial Serial Peripheral Interface (DSPI)*: - Removed all the note that refer to Rx Mask. - Removed DSPIx_CTAR6 and DSPIx_CTAR7 register. - Added following tables: *Table 242*, *Table 243*, *Table 244*, *Table 245*, *Table 246*, *Table 247*, *Table 248*, *Table 249*, *Table 250*, *Table 251*, *Table 252*, *Table 253*, *Table 254*. *Section 20.4 Features*: Replaced “Eight clock and transfer attribute registers“ with “Six clock and transfer attribute registers“.

Table 433. Document revision history (continued)

Date	Revision	Changes
19-Feb-2009	2 (continued)	*Section DSPI Module Configuration Register (DSPIx_MCR)*: Removed CLR_TXF and CLR_RXF fields from DSPIx_MCR register. *Chapter 21 LIN Controller (LINFlex)*: Replaced with a new chapter. *Chapter 22 FlexCAN*: - Removed “[Ref.1]”. - *Section 22.1.2 FlexCAN module features*. - Added bullet “Hardware cancellation on Tx message buffers.“ . - Removed note. - *Table 326*: Replaced the footnote with new one. - *Section 22.3.3 Rx FIFO structure*: *Table 274* fixed the offset value. - Updated *Section Module Configuration Register (MCR)* (MAXMB note). - Updated *Section Error and Status Register (ESR)* (bit numbers in first paragraph). - Fixed information about the number of frames accumulated in the FIFO to generate a warning interrupt, which is 5. (Affected sections: *Section Interrupt Flags 1 Register (IFLAG1)* and *Section 22.4.8 Rx FIFO*). - *Section 22.4.2 Local priority transmission*: added. *Chapter 24 Inter-Integrated Circuit Bus Controller Module (I2C)*: - Replaced “I2C_DMA“ “I2C“ through whole chapter. - *Section 24.1.1 Overview*: Replaced the capacitance value from “400pF“ to “50pF“. - *Table 329*: Removed Mode column. *Chapter 25 Configurable Enhanced Modular IO Subsystem (eMIOS)*: Replaced with a new chapter. *Chapter 26 Analog-to-Digital Converter (ADC)*: Replaced with a new chapter. *Chapter 27 Cross Triggering Unit (CTU)* - *Table 381*: Corrected eMIOS channel assignment on CTU inputs. *Chapter 29 System Status and Configuration Module (SSCM)*: - *Section System Memory Configuration Register (MEMCONFIG)*: Updated register definition. - Removed section “Initialization/Application Information“.

Table 433. Document revision history (continued)

Date	Revision	Changes
19-Feb-2009	2 (continued)	*Chapter 30 Wakeup Unit (WKPU)*: - Related Bit12 and Bit13 to the wakeup line respectively 19 and 18 of the following register: WISR, IRER, WRER, WIREER, WIFEER, WIFER, WIPUER. - WKUP line 0: Previously = RTC/API; Now = API. - 'Old'WKUP line 1 -> Now WKUP line 19. - 'New' WKUP line 1 -> RTC. - Replaced “NMI[0]“ whti [NMI]. - Moved the note of “Interrupt Vector 2”, this note is valid only for PF[13], PG[3] and PG[5]. - *Figure 421*: Replaced “0-18” instead “0-19”. - *Figure 433*: updated in according to the previous change. - *Section 30.2 Features*: - Updated the “External wakeup/interrupt support” list to explain that system interrupt vectors are 3. - Updated the “On-chip wakeup support“ list to explain that wakeup spurces are 2. - *Section 30.4.1 Memory map*: *Table 406* Removed redundant rows. *Chapter 31 Periodic Interrupt Timer (PIT)*: - *Figure 434*: Replaced “Timer 3” with “Timer 5”. *Chapter 33 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)*: - Added “/Autonomous Periodic Interrupt“ in the title. - *Figure 444*: Updated to explain that “RTC Rollover wakeup“ and “RTC cnt_or_rlovr“ are not connected on SPC560Bx and SPC560Cx. - *Figure 33.4*: Removed section “Test mode“. - Removed section “External Signal Description“. *Chapter 34 Voltage Regulators and Power Supplies*: Aligned the electrical value with datasheet
08-May-2009	3	Improved the formatting through whole document. *Chapter 3 Clock description*: fixed the issue about the bit numbering of all registers. *Chapter 10 Interrupt Controller (INTC)*: - *Figure 99*: Updated. *Chapter 25 Configurable Enhanced Modular IO Subsystem (eMIOS)*: - *Figure 307*: Updated. *Chapter 26 Analog-to-Digital Converter (ADC)*: Updated the following tables: *Figure 365*, *Figure 368*, *Figure 376*, *Figure 377*, *Figure 378*, *Figure 379*, *Figure 380*, *Figure 381*, *Figure 384*, *Figure 385*, *Figure 388*, *Figure 391*, *Figure 392*, *Figure 393*. *Chapter 29 System Status and Configuration Module (SSCM)*: Updated the following tables: *Figure 419*, *Figure 420*, *Table 404*.

Table 433. Document revision history (continued)

Date	Revision	Changes
08-May-2009	3 (continued)	*Chapter 30 Wakeup Unit (WKPU)*: Updated the following tables: *Figure 425*, *Table 409*, *Figure 426*, *Table 410*, *Figure 427*. *Chapter 33 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)*: Updated the following tables: *Table 435*, *Figure 449*, *Figure 450*, *Table 437*. *Chapter 34 Voltage Regulators and Power Supplies*: Updated *Figure 451*.
13-Jan-2010	4	*Chapter 1 Overview* Minor editorial and formatting changes Updated SPC560Bx and SPC560Cx series block diagram *Section 1.1.3 Chip-level features*: Changed eMIOS-lite to eMIOS *Section 1.3 Developer support*: Added footnote defining AUTOSAR Memory map: – Changed Periodic Interrupt Timer (PIT/RTI) to Periodic Interrupt Timer (PIT) – Changed CTU-LITE to CTU – Changed SRAM size from 32 KB to 48 KB *Chapter 2 Signal description* Minor editorial and formatting changes *Section 2.2 Package pinouts*: Inverted the order of *Figure 2* and *Figure 3* 208 MAPBGALBGA208 configuration: Changed description for ball H1 from NC to VSS_HV *Section 2.3 Pad configuration during reset phases*: Added BAM function ABS[0] to PA[8] Voltage supply pin descriptions: Added ball H1 to VSS_HV pins Functional port pin descriptions: – Added a footnote regarding “I/O Direction” column – Replaced GPIO[20] with GPI[20] – Changed GPI[21] to GPIO[21] – Changed the “Reset config” of PB[7] and PB[8] to Tristate – Changed “Pad Type“ from S to M in 27 pads – Changed pad type from S to M on port pin PE[7] *Chapter 3 Clock description* Editorial and formatting changes *Section 3.5, “Memory Map and Register Definition”*, *Section 3.3.3 Register description* Added ‘Location’ column to MC_CGM Register Description; added clock domain information to clock source selection register descriptions *Section 3.5 Slow internal RC oscillator (SIRC) digital interface*: Replaced all LPRC occurrences with SIRC

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Section Normal mode*: Replaced “CR“ with “CR.NDIV“ and “LDF” with “NDIV“ Fast External Crystal Oscillator Control Register (FXOSC_CTL) field descriptions: Updated description of EOCV[7:0] FMPLL block diagram: Added footnote to DIV2 FMPLL memory map: Updated access types CR field descriptions: Updated description of field EN_PLL_SW Progressive clock switching on pll_select rising edge: Updated column header titles Added figure “FMPLL output clock division flow during progressive switching” *Chapter 5 Mode Entry Module (MC_ME)* *Section 5.3 Memory Map and Register Definition*: Minor editorial and formatting changes; RESET Mode Configuration Register (ME_RESET_MC): Corrected title of bitmap, *Section 5.4.1 Mode Transition Request*, *Section RESET Mode*, *Section STANDBY0 Mode*: Added ‘Location’ column to MC_ME Register Description; added note for S_MTRANS polling; cleaned up MC_ME Mode Diagram; added details to RESET mode description; added details of booting from backup RAM on STANDBY0 exit *Chapter 6 Boot Assist Module (BAM)* Minor editorial and formatting changes to improve readability Updated oscillator naming Removed all references to “autobaud” and to ABD field of SSCM_STATUS register (autobaud feature not supported by device) *Section 6.3.2 Reset Configuration Half Word Source (RCHW)*: Changed offset from 0x02 to 0x00 *Section 6.3.3 Single chip boot mode*: Added a footnote BAM memory organization: Added column header “Parameter” Updated Fields of SSCM STATUS register used by BAM *Section BAM resources*: Updated list of MCU resources *Section Download and execute the new code*: Removed optional first step No. 0 (step concerned send/receive message for autobaud rate selection) Updated Serial boot mode – baud rates Updated System clock frequency related to external clock frequency Reset Configuration Half Word (RCHW): Changed reset value for all fields: was 0; is 1 Updated *Section Download 64-bit password and password check* *Chapter 7 Reset Generation Module (MC_RGM)* Added ‘Location’ column as navigational aid to MC_RGM Register Description *Chapter 8 System Integration Unit Lite (SIUL)* Editorial and formatting changes to improve readability Updated SIUL signal properties Updated SIUL memory map Updated register descriptions *Section 8.6.2 General purpose input and output pads (GPIO)*: Updated number of interrupt vectors and number of external interrupts

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Chapter 9 Power Control Unit (MC_PCU)* Added 'Location' column as navigational aid to MC_PCU Register Description *Chapter 10 Interrupt Controller (INTC)*: Editorial and formatting changes *Chapter 11 e200z0h Core* Minor editorial and formatting changes Updated e200z0h block diagram *Section e200z0h system bus features*: Added footnotes *Chapter 12 Peripheral bridge (PBRIDGE)* Editorial and formatting changes, including chapter title change Replaced "AIPS" with "peripheral bridge", or "PBRIDGE" where appropriate, throughout chapter Peripheral bridge interface: Updated PBRIDGE1 peripheral names Updated *Section 12.1.4 Modes of operation* *Chapter 13 Crossbar Switch (XBAR)* Minor editorial and formatting changes Updated XBAR block diagram *Chapter 14 Memory Protection Unit (MPU)* Minor editorial and formatting changes Updated MPU block diagram Updated *Section 14.2.2 Register description* to include adding bit numbers to field names and changing field bit numbers format to LSB=0 where needed MPU memory map: Removed MPU_EAR3 and MPU_EDR3 MPU Error Address Register, Slave Port n (MPU_EARn): Removed MPU_EAR3 content MPU Error Detail Register, Slave Port n (MPU_EDRn): Removed MPU_EDR3 content MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0): Replaced asterisks with '0' as reset value for bits 27:31 MPU_RGDn.Word0 field descriptions: Replaced SRTADDR[31:0] with SRTADDR[26:0] MPU_RGDn.Word1 field descriptions: Replaced ENDADDR[31:0] with ENDADDR[26:0] *Chapter 15 Error Correction Status Module (ECSM)* Editorial and formatting changes; repaired cross-reference links Replaced AIPS with "peripheral bridge" or "PBRIDGE" *Section 15.4.2 Register description*: Applied LSB=0 to field internal bit numbers ECSM 32-bit memory map: Added ECSM base address *Section Processor Core Type (PCT) register*: Added reset values to bitmap *Section Revision (REV) register*: Added reset values to bitmap *Section IPS Module Configuration (IMC) register*: Added reset values to bitmap

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Section Miscellaneous User-Defined Control Register (MUDCR)* – Updated bit numbers and field descriptions – Updated text following field description table *Section ECC Configuration Register (ECR)*: Removed paragraph about reporting of single-bit memory corrections Updated ECC Configuration (ECR) field descriptions *Section ECC Error Generation Register (EEGR)*: Removed paragraph about enabling of error generation modes *Section ECC Error Generation Register (EEGR)*: Replaced “for the ECC Configuration Register definition” with “for the ECC Error Generation Register definition” in sentence above bitmap Updated ECC Error Generation (EEGR) field descriptions *Chapter 16 IEEE 1149.1 Test Access Port Controller (JTAGC)* Editorial and formatting changes; repaired cross-references *Section 16.1 Introduction*: Removed paragraph about IEEE 1149.7 e200z0 OnCE Register Addressing: Replaced ‘Shared Nexus Control Register (SNC)’ with ‘Reserved’ (SNC register not implemented on this device) *Chapter 17 Nexus Development Interface (NDI)* Editorial and formatting changes NDI Implementation Block Diagram: Replaced PPC with CPU Nexus Debug Interface Registers: – Added ‘Location’ column as navigational aid – Removed Client Select Control (CSC) Register (CSC register not implemented on this device) – Updated register names – Removed sentence referencing device MPC5516 from footnote *1* Nexus Device ID (DID) Register bitmap: Changed reset value for field MIC—was 0xE, is 0x20 PCR field descriptions: Updated description of MCKO_DIV[2:0] Updated *Section 17.7.3 Programmable MCKO Frequency* *Section 16.7.4, “Nexus Messaging”*: Removed sentence referencing Client Select Control Register *Section 16.7.6.1, “EVTI Generated Break Request”*: Removed sentence referencing Shared Nexus Control (SNC) Register (SNC register not implemented on this device) *Chapter 18 Static RAM (SRAM)* Editorial and formatting changes, including modification of chapter title *Section 18.3 Register memory map*: Replaced “32 KB” with “48 KB” in first sentence *Chapter 19 Flash memory* Editorial and formatting changes *Section 19.1 Introduction*: Replaced 544 Kbyte with 512 Kbyte Flash memory architecture: Replaced “EEE” with “EEPROM emulation” *Section 19.2 Code Flash*: Changed title and content to replace “Program Flash” with “Code Flash”

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	Updated *Section 19.2.1 Introduction* *Section 19.2.2 Main features*: Removed bullet “Usable as main Code Memory” Updated *Section 19.2.3 Block diagram* Updated *Section Flash module sectorization* Updated *Section 19.2.6 Module Configuration Register (MCR)* Updated *Section 19.2.8 High address space Block Locking register (HBL)* Updated *Section 19.2.11 High address space Block Select register (HBS)* ADR field descriptions: Removed the phrase “if the device is configured to show this feature” in the AD22-3 description *Section 19.2.13 Bus Interface Unit 0 register (BIU0)*: – Removed sentence “The availability of this register is device dependent”. – Updated BIU0 field descriptions *Section 19.2.14 Bus Interface Unit 1 register (BIU1)*: – Removed sentence “The availability of this register is device dependent”. – Updated BIU1 field descriptions *Section Non-volatile Bus Interface Unit 2 register (NVBIU2)*: – Removed sentence “The availability of this register is device dependent”. – Updated BIU2 field descriptions *Section 19.2.16 User Test 0 register (UT0)*: Modified first sentence Non-volatile private censorship PassWord 0 register (NVPWD0): Changed delivery value 0xXXXXXXXX to 0xFFFF_FFFF Non-volatile private censorship PassWord 1 register (NVPWD1): Changed delivery value 0xXXXXXXXX to 0xFFFF_FFFF NVSCI0 field descriptions: Replaced “or NVSCI1 = NVSCI0” with “or NVSCI1 != NVSCI0” in fields SC and CW NVSCI1 field descriptions: – Replaced “SC32-16: *Serial Censorship control word 32-16* (Read/Write)” with “SC[31:16]: *Serial Censorship control word 31-16* (Read/Write)” – Replaced “CW32-16: *Censorship control Word 32-16* (Read/Write)” with “CW[31:16]: *Censorship control Word 31-16* (Read/Write)” – Replaced “or NVSCI1 = NVSCI0” with “or NVSCI1 != NVSCI0” in fields SC and CW *Section 19.2.28 Non-volatile User Options register (NVUSRO)*: – Removed sentence “The availability of this register is device dependent”. – Updated *Table 198* Updated *Table 199* *Section 19.3.14 User Test 0 register (UT0)*: Modified first sentence *Section 19.4.1 Introduction*: Replaced AIPS-Lite with PBRIDGE PFCR0 field descriptions: Modified field descriptions for BK0_APC, BK0_WWSC and BK0_RWSC PFCR1 field descriptions: Modified field descriptions for BK1_APC, BK1_WWSC and BK1_RWSC *Section 19.5.13 Timing diagrams*: Reformatted and rescaled timing diagrams to improve readability and alignment of content

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Chapter 20 Deserial Serial Peripheral Interface (DSPI)* Editorial and formatting changes Removed all references to DSI (Deserial Serial Interface) and CSI (Combined Serial Interface) (device does not implement DSI and CSI) Removed all references to DMA (DSPI does not support DMA in this device) DSPI Module Configuration Register (DSPIx_MCR): Replaced the reset value of MDIS bitfield with '1'. Made same modification on DSPIx_MCR field descriptions. *Section 20.3 Overview*: Replaced "DSPI 0, DSPI 1 and DSPI 2" with "DSPI 0 – DSPI 5" *Section 20.4 Features*: Removed bullet "Supports all functional modes from QSPI subblock of QSMCM (MPC500 family)" *Section 20.5 Modes of operation*: Updated to add External Stop mode *Section 20.6.2 Signal names and descriptions*: Formatted all CS signals as "CSn_x" *Section 20.7 Memory map and register description*: Removed all DMA requests content *Section DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn)*: Changed number of clock and transfer attribute registers from eight to six *Section DSPI Status Register (DSPIx_SR)*: Modified first paragraph DSPI detailed memory map: Added 'Location' column as navigational aid Baud rate computation example: Changed f_{SYS} from 100 MHz to 64 MHz and updated baud rate accordingly *Section 20.8 Functional description*: Removed all DMA requests content and eDMA controller content *Section 20.8.1 Modes of operation*: Updated to add External Stop mode *Section 20.9.1 How to change queues*: Modified list of events: Was 1–11, is 1–7 Updated *Section 20.9.3 Delay settings* *Chapter 21 LIN Controller (LINFlex)* Editorial and formatting changes Updated *Section 21.2.1 LIN mode features* Added names for *Example 15* and *Example 16* Replaced LINFlex memory map *Section 21.7.2 Register description* – Aligned hexidecimal reset values to reset values shown in bitmaps where necessary – Aligned bit numbering in register field description tables to numbering in register bitmaps where necessary LIN control register 1 (LINCR1): – Changed reset value from 0x0082_0000 to 0x0000_0082 – Changed access from w1c to R/W for fields CCD, CFD, LASE, AWUM, MBL[0:3], BF, SFTM, LBKM, MME, SBDT, RBLM, SLEEP and INIT LIN interrupt enable register (LINIER): – Updated LSIE field description – Changed access from w1c to R/W for fields SZIE, OCIE, BEIE, CEIE, HEIE, FEIE, BOIE, LSIE, WUIE, DBFIE, DBEIE, DRIE, DTIE and HRIE *Section LIN status register (LINSR)*: Updated LINS field description; changed access from w1c to read-only for field RPS

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Section LIN error status register (LINESR)*: Updated SZF field description UART mode control register (UARTCR): Changed access from w1c to R/W for fields RXEN, TXEN, OP, PCE, WL and UART UARTSR field descriptions: Added footnote *1* LIN timeout control status register (LINTCSR): Changed access from w1c to R/W for fields LTOM, IOT and TOCE LIN output compare register (LINOCR): Changed access from w1c to R/W for OCx[0:7] *Section LIN timeout control status register (LINTCSR)*: Updated HTO field description LIN fractional baud rate register (LINFBRR): Changed access from w1c to R/W for DIV_F[0:3] LIN integer baud rate register (LINIBRR): Changed access from w1c to R/W for DIV_M[0:12] LIN checksum field register (LINCFR): Changed access from w1c to R/W for CF[0:7] LIN control register 2 (LINCR2): – Changed access from w1c to R/W for fields IOBE and IOPE – Changed access from w1c to write-only for fields WURQ, DDRQ, DTRQ, ABRQ and HTRQ *Section Buffer identifier register (BIDR)*: Updated CCS field description; changed access from w1c to R/W for fields DIR and CCS Buffer data register LSB (BDRL): Changed access from w1c to R/W for DATAx[0:7] *Section Identifier filter enable register (IFER)*: Updated description of FACT[0:7]; added IFER[FACT] configuration table *Section Identifier filter match index (IFMI)*: Replaced IFMI[0:3] with IFMI[0:4] *Section Identifier filter mode register (IFMR)*: Replaced IFM[0:3] with IFM[0:7]; added IFMR[IFM] configuration table; changed register access from User read-only to User read/write; changed access from read-only to R/W for IFM[0:7] *Section Identifier filter control register (IFCR2n)*: Amended address offsets; changed access from w1c to R/W for fields DIR and CCS *Section Identifier filter control register (IFCR2n + 1)*: Amended address offsets; changed access from w1c to R/W for fields DIR and CCS Register map and reset values: – Updated bits of IFMI and IFMR – Amended address offsets for IFCR2n and for IFCR2n+1 Added *Section Clock gating* to *Section 21.8.1 UART mode* *Section 21.8.2 LIN mode*: Added footnote regarding slave mode Updated *Section Data reception (transceiver as subscriber)* Updated *Section Data discard* Updated *Section Data transmission (transceiver as publisher)* Updated *Section Data reception (transceiver as subscriber)* Updated *Section Data discard* Filter configuration—register organization: Replaced ID5:0 with ID[0:5] Added *Section Clock gating* Updated *Section 21.8.3 8-bit timeout counter* Header and response timeout: Updated arrows for OC_{Header}, $OC_{Response}$ and OC_{Frame}

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Chapter 22 FlexCAN* Editorial and formatting changes, including removing all references to consulting a device user guide Removed any reference to the FlexCAN wake-up interrupt Module memory map: Added ‘Location’ column as navigational aid Updated *Section Module Configuration Register (MCR)* Updated *Section Control Register (CTRL)* Added text in *Section Rx Global Mask (RXGMASK)*, *Section Rx 14 Mask (RX14MASK)* and *Section Rx 15 Mask (RX15MASK)* referring to *Section 22.4.8 Rx FIFO* *Section Message buffer lock mechanism*: Added a note Error and Status Register (ESR) field description: Updated titles of bits TX_WRN and RX_WRN CAN standard compliant bit time segment settings: specified that it refers to the official CAN specification *Chapter 23 CAN sampler* Formatting and editorial changes Updated oscillator naming (“16 MHz fast internal RC oscillator”) Updated *Section 23.3 Register description* *Section 23.4 Functional description*: Removed section “Selecting the Rx port” (information already exists in register field description in *Table 327*) *Section 23.4.2 Baud rate generation*: Replaced BRP bits 5:1 with BRP[4:0] CAN sampler register map: Updated field descriptions *Chapter 24 Inter-Integrated Circuit Bus Controller Module (I2C)* Formatting and editorial changes throughout, including harmonizing register names Module Memory Map: Added ‘Location’ column as navigational aid IBSR field descriptions: Removed comment “Check w/design if this is the case (only TCF)” from description of field IBIF *Interrupt Description*: Removed comment “To be checked” from Byte Transfer condition *Chapter 25 Configurable Enhanced Modular IO Subsystem (eMIOS)* Organizational, editorial and formatting changes, including changing ‘$’ to ‘0x’ throughout *Section 25.1.2 Features of the eMIOS module*: Removed “identical” from first bullet in list Modified *Section 25.1.1 Overview of the eMIOS module* Removed “identical’ from first bullet in features list Channel configuration: – Modified eMIOS block numbering—Was eMIOS_A and eMIOS_B, is eMIOS_0 and eMIOS_1 – Corrected position of horizontal arrow between Counter Bus_B and Ch1 in eMIOS_0 – Added GPIO to diagram key Updated *Section Channel mode selection* *Section 25.3 Memory map and register description*: Harmonized register naming and added location columns to memory map tables

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	eMIOS Module Configuration Register (EMIOSMCR): Changed reset value of MDIS to '0' EMIOSMCR field descriptions: Corrected table title EMIOSOUDIS register field descriptions: Replaced OU31:OU0 with OU27:OU0 Updated *Section eMIOS UC Control (EMIOSC[n]) Register* UC BSL bits: Added "Channels 24 to 27: counter bus[E]" to selected bus for field value '01' EMIOSS[n] register field descriptions: Updated FLAG field description *Section 25.4 Functional description*: Changed the number of channel types; was three, is five Updated *Section Coherent accesses* Unified Channel block diagram: – Changed ips_wda to ips_wdata[0:31] – Changed uc_rd_d to uc_rd_data[0:31] – Changed ips_ad to ips_addr[27:29] *Chapter 26 Analog-to-Digital Converter (ADC)* Formatting changes *Section 26.1.1 Device-specific features*: – Replaced MA[0:2] with MA[2:0] – Removed 1.2 V from presampling options Updated ADC implementation diagram Updated *Section 26.2 Introduction* *Section Normal conversion*: Minor editorial change *Section Start of normal conversion*: Minor editorial change Updated second paragraph in *Section 26.3.2 Analog clock generator and conversion timings* Updated *Section 26.3.3 ADC sampling and conversion timing* Updated *Section Presampling channel enable signals* Updated Presampling voltage selection based on PREVALx fields Updated *Section 26.3.7 Interrupts* Main Configuration Register (MCR) field descriptions: Updated description for field OWREN Main Status Register (MSR) field descriptions: Updated values for ADCSTATUS[0:2] (and removed stand-alone description table for this field) Watchdog Threshold Interrupt Status Register (WTISR) field descriptions: Changed "corresponds to the interrupt generated " to "corresponds to the status flag generated" in both bit descriptions Presampling Control Register (PSCR) field descriptions: Updated descriptions for PREVAL fields *Section 26.4.6 Conversion timing registers CTR[0..2]*: Restored OFFSHIFT field Channel Data Register (CDR[0..95]) field descriptions: – Updated description for field OVERW – Added value '11' to field RESULT[0:1]

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Chapter 27 Cross Triggering Unit (CTU)* Minor editorial and formatting changes CTU register map: Added 'Location' column and base address Trigger source: Corrected eMIOS channel assignment on CTU inputs Amended end of *Section 27.4.1 Event Configuration Register (CTU_EVTCFGRx) (x = 0...63)* *Chapter 28 Safety* Editorial and formatting changes *Section Overview*: Replaced AIPS with PBRIDGE *Section 28.2 Software Watchdog Timer (SWT)*: Replaced "Software" by "System" Replaced "system watchdog" with "software watchdog" throughout chapter SWT memory map: Added 'Location' column as navigational aid *Chapter 29 System Status and Configuration Module (SSCM)* Formatting and editorial changes *Section 29.2.2 Register description*: Updated introduction and applied LSB=0 numbering to field bit numbers where needed Module memory map: Added 'Location' column as navigational aid Error Configuration (ERROR) field descriptions: Replaced "AIPS" with "PBRIDGE" in RAE field description System Memory Configuration Register (MEMCONFIG) field descriptions: Modified descriptions of fields PRSZ and PVLB to replace "Program Flash" and "Instruction Flash" with "Code Flash" *Chapter 30 Wakeup Unit (WKPU)* Editorial and formatting changes Updated *Section 30.1 Overview* Updated Wakeup unit block diagram WKPU memory map: Added 'Location' column as navigational aid Interrupt vector 1: Updated PB[3] Interrupt vector 2: Updated PG[3] and PG[5] Wakeup/Interrupt Status Flag Register (WISR): Updated footnote Updated WISR field descriptions Interrupt Request Enable Register (IRER): Updated footnote Wakeup Request Enable Register (WRER): Updated footnote Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER): Updated footnote Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER): Updated footnote Wakeup/Interrupt Filter Enable Register (WIFER): Updated footnote Wakeup/Interrupt Pullup Enable Register (WIPUER): Updated footnote *Section 30.5.3 External wakeups/interrupts*: Replaced "supports up to two interrupt vectors" with "supports up to three interrupt vectors"

Table 433. Document revision history (continued)

Date	Revision	Changes
13-Jan-2010	4 (continued)	*Chapter 31 Periodic Interrupt Timer (PIT)* Editorial and formatting changes Replaced PIT_RTI with PIT throughout document Tables PIT memory map and Timer Channel n: Added “Location” column as navigational aid *Section 31.5.1 Example configuration*: Removed RTI lines from code *Chapter 32 System Timer Module (STM)*: No content changes *Chapter 33 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)* Editorial and formatting changes, including updating and harmonizing oscillator names *Section 33.5 Register descriptions*: Added register map; added key to register fields *Chapter 34 Voltage Regulators and Power Supplies* Formatting and editing changes Updated figure Updated *Section 34.1.1 High power regulator (HPREG)* *Section 34.3 Power domain organization*: Modified number of power domains; was two, is three *Appendix A: Register Under Protection*: No content changes *Appendix B: Register Map* Minor editorial and formatting changes Module base addresses: – Changed Periodic Interrupt Timer (PIT/RTI) to Periodic Interrupt Timer (PIT) – Changed CTU-LITE to CTU Detailed register map: – Changed register name PIT_RTI Control to PIT_Control – Changed Periodic Interrupt Timer (PIT/RTI) to Periodic Interrupt Timer (PIT) – Changed CTU-LITE to CTU – Updated description of RSER – Replaced “Program Flash A Configuration” with “Code Flash A Configuration” – Replaced registers IFER, IFMI, IFMR, IFCR2n and IFCR2n+1 with “Reserved” for LINFlex modules 1, 2 and 3

Table 433. Document revision history (continued)

Date	Revision	Changes
31-Mar-2010	5 (continued)	Internal release. *Chapter 1 Overview* “SPC560Bx and SPC560Cx series block diagram”figure: – Added “Interrupt request with wake-up functionality” as an input to the WKPU block. *Chapter 2 Signal description* “Functional port pin description” table: – Improved the footnote regarding JTAGC pins in order to explain when the device get incompliance with IEEE 1149.1-2001. – Added a footnote concerning the family compatibility. – Footnote 11: Replaced ”SPC560B44L3 and SPC560B44L” with “SPC560B40L3 and SPC560B40L5”. *Chapter 3 Clock description* “SPC560Bx and SPC560Cx System Clock Generation” figure: – Changed the dividers from 1 to 15 to 1 to 16 of the system clock selectors. “Progressive clock switching” section: – Revised. – Added “Progressive clock switching scheme” figure. – Update definition of en_pll_sw bit filed on Control Register. “Slow external crystal oscillator (SXOSC) digital interface“ section: – Interrupt functionalities are not available on SXOSC. *Chapter 6 Boot Assist Module (BAM)* “Hardware configuration to select boot mode“ table: – Renamed the flag "Standby-RAM Boot Flag" to "BOOT_FROM_BKP_RAM". “Download 64-bit password and password check“ section: – Added note about password management. “Boot from FlexCAN“ section: – Added note about the distirb provided by CAN traffic. *Chapter 10 Interrupt Controller (INTC)* Replaced INTC_PSR121 with “INTC_PSR147 Updated “INTC Priority Select Registers“ and “INTC Priority Select Register Address Offsets“ table in according to “Interrupt Vector Table“ table *Chapter 16 IEEE 1149.1 Test Access Port Controller (JTAGC)* “External Signal Description“ section: – Emphasized when the device get incompliance with IEEE 1149.1-2001. *Chapter 17 Nexus Development Interface (NDI)* “Ownership Trace“ section: – Added it. *Chapter 19 Flash memory* – Updated delivery values of NVPWD0 and NVPWD1 for Code Flash. – Revised the “Margin read“ section for both Flash. – Replaced “Margin Mode“ with “Margin Read”.

Table 433. Document revision history (continued)

Date	Revision	Changes
31-Mar-2010	5 (continued)	*Chapter 20 Deserial Serial Peripheral Interface (DSPI)* “DSPIx_MCR register“: – Included Bit fields CLR_TXF and CLR_RXF. *Chapter 21 LIN Controller (LINFlex)* “LINTCSR“ register: – Updated the reset value. *Chapter 22 FlexCAN* “Control Register (CTRL) field description” table: – Sorted correctly the bit fields. *Chapter 26 Analog-to-Digital Converter (ADC)* “Threshold Control” Register: – Removed THRINV field. Decode Signals Delay Register (DSDR): – Update the description. “Max AD_clk frequency and related configuration settings“ table: – Adeed a footnote. *Chapter 28 Safety* “SWT_CR” Register: – Added the field “KEY”. *Chapter 33 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)* “RTCC“ Register: – Updated the APIVAL description.

Table 433. Document revision history (continued)

Date	Revision	Changes
05-Aug-2010	6	*Chapter 2 Signal description* 100 LQFP pinout and 144 LQFP pinout: • Removed alternate functions 208 MAPBGA pinout: • OSC32K_XTAL at R9 changed to XTAL32 • OSC32K_EXTAL at T9 changed to EXTAL32 *Chapter 3 Clock description* Revised “Progressive clock switching” section. Added “Progressive clock switching” scheme figure Update definition of en_pll_sw bit filed on Control Register Interrupt functionalities are not available on SXOSC *Chapter 6 Boot Assist Module (BAM)* Added notes in the following section: Download 64-bit password and password check Download data Execute code *Chapter 8 System Integration Unit Lite (SIUL)* Clarified description of I/O pad function in overview section Clarification: Not all GPIO pins have both input and output functions Replaced parallel port register sections (PGPDO, PGPDI, and MPGDO), clarifying register function and bit ordering Editorial updates throughout chapter *Chapter 16 IEEE 1149.1 Test Access Port Controller (JTAGC)* Changed the code values for ACCESS_AUX_TAP_TCU and ACCESS_AUX_TAP_NPC in the “JTAG Instructions” table *Chapter 19 Flash memory* Added a note in thr “Censorship password register” sections Added information on RWW-Error during stall-while-write in the “Module Configuration Register (MCR)”

Table 433. Document revision history (continued)

Date	Revision	Changes
05-Aug-2010	6 (continued)	*Chapter 26 Analog-to-Digital Converter (ADC)* Updated following section: • Overview • Introduction • Injected channel conversion • Abort conversion • ADC CTU (Cross Triggering Unit) • Presampling Updated following registers: • CEOCFR • CIMR • WTISR • DMAR • PSR • NCMR • JCMR • CDR • CWSEL • CWENR • AWORR Inserted "CTU triggered conversion" in the conversion list of "Functional description" section Replaced generic “system clock” with “peripheral set 3 clock” added information about “ADC_1” in the “ADC sampling and conversion timing” section Moved CWSEL, CWENR and AWORR register within “Watchdog register“ section Inserted a footnote about OFFSHIFT field in the CTR register Changed the access type of DSDR in "read/write" Updated the DSD description in the DSDR field description table *Chapter 27 Cross Triggering Unit (CTU)* Replaced “Channel number value mapping” table with “CTU-to-ADC Channel Assignment” table Removed “Control Status Register (CTU_CSR)” because the interrupt feature is not implemented. Cross Triggering Unit block diagram: trigger output control and output signals removed Main Features section: Removed “Maskable interrupt generation whenever a trigger output is generated”. Feature not implemented.
01-Oct-2011	7	Chapter Throughout Editorial changes and improvements (including reformatting of memory maps, register figures, and field descriptions to a consistent format). Rearranged the chapter order. Chapter Preface Added this chapter. Chapter Introduction Changed the chapter title (was “Overview”, is “Introduction”). Renamed “Introduction” to “The SPC560Bx and SPC560Cx microcontroller family” and revised the section. Renamed “Feature summary” to “Feature details”. Moved the “Memory map” section to its own separate chapter. Deleted the duplicate device-comparison tables. In the Packages section, added a line for the 64-pin LQFP.

Table 433. Document revision history (continued)

Date	Revision	Changes
01-Oct-2011	7 (continued)	Chapter Memory Map Added this chapter (content previously contained in the Overview chapter). Changed “Test Sector Data Flash Array 0” to “Data test sector”. Revised the numbers in the “Code Flash Sector” entries. Changed “Flash Shadow Sector” to “Code Flash Shadow Sector”. Changed “Code Flash Array 0 Test Sector” to “Code Flash Test Sector”. Revised the numbers in the “Data Flash Array” entries. Consolidated multiple adjacent reserved rows into single rows. Chapter Signal Description Added the 64-pin LQFP package figure. In the “Voltage supply pin descriptions” table, added pin 6 to the entry for VSS_HV in the 64-pin package. Changed “Functional ports A, B, C, D, E, F, G, H” to “Functional ports”. In the “Functional ports” table, changed ANP[0]–ANP[15] to GPI[0]–GPI[15]. Chapter Safety Migrated the chapter contents to the “Register Protection” and “SWT” chapters. Chapter Microcontroller Boot Added this chapter. Chapter Clock Description Replaced the “SPC560Bx and SPC560Cx system clock generation” figure with the version present in Rev. 5 of the SPC560Bx and SPC560Cx reference manual. Fast external crystal oscillator (FXOSC) digital interface section: Changed the sentence from “The FXOSC digital interface controls the 4–40 MHz fast external crystal oscillator (FXOSC).” to “The FXOSC digital interface controls the operation of the 4–40 MHz fast external crystal oscillator (FXOSC).” Truth table of crystal oscillator table: Replaced "ME_GS.S_XOSC" with “ME_xxx_MC[FXOSCON]", replaced “FXOSC_CTL.OSCBYP” with “FXOSC_CTL[OSCBYP]” Slow external crystal oscillator (SXOSC) digital interface section: Changed the sentence from “The SXOSC digital interface controls the 32 KHz slow external crystal oscillator (SXOSC).” to “The SXOSC digital interface controls the operation of the 32 KHz slow external crystal oscillator (SXOSC).” SXOSC truth table: Replaced "S_OSC” with “OSCON" Renamed the figure title from “RC Oscillator Control Register (RC_CTL)” to “FIRC Oscillator Control Register (FIRC_CTL)” Renamed the table title from “RC Oscillator Control Register (RC_CTL) field descriptions” to “FIRC Oscillator Control Register (FIRC_CTL) field descriptions” In the FXOSC_CTL figure, added footnotes to clarify the access to the OSCBYP and I_OSC fields. Deleted the “CMU register map” section. Added notes for clarifying field access to the following registers – FXOSC_CTL – SXOSC_CTL – CMU_CSR

Table 433. Document revision history (continued)

Date	Revision	Changes
01-Oct-2011	7 (continued)	Revised the SXOSC_CTL section. In the SIRC “Functional description” section, revised the information of SIRC output frequency trimming. In the FIRC “Functional description” section, revised the information of FIRC output frequency trimming. In the FIRC_CTL section, deleted the FIRCON_STDBY field. Revised the reset values in the FMPLL CR. Revised the SIRC_CTL[SIRCTRIM] field description. Revised the FIRC_CTL[FIRCTRIM] field description. Changed STANDBY0 to STANDBY. In the FMPLL features, changed “SSCG” to “frequency modulation”. In the FMPLL functional description, added the “FMPLL lookup table” table. In the CMU introduction, changed “towards the mode” to “towards the MC_ME”. In the CMU introduction, deleted the “CMU block diagram” figure. In the CMU Introduction section, changed “clock management unit” to MC_CGM. Chapter Mode Entry Module Changed “WARNING” to “CAUTION”. Changed HALT0 to HALT. Changed STOP0 to STOP. Changed STANDBY0 to STANDBY. Added the “Peripheral control registers by peripheral” table. In the ME_<mode>_MC[DFLAON] field description, added a note about configuring reset sources as long resets. Chapter Reset Generation Module Changed STANDBY0 to STANDBY. Revised the chapter to reflect the fact that the RGM_DEAR and RGM_DERD registers are always read-only. In the “External Reset” section, changed “In this case, the external reset is forced low by the product until the beginning of PHASE3” to “In this case, the external reset is asserted until the end of PHASE3”. Revised the RGM_FEAR[AR_CMU_OLR] field description. Revised the RGM_FES[F_CORE] field description. Changed “core reset” to “debug control core reset”. Chapter Power Control Unit Changed HALT0 to HALT. Changed STOP0 to STOP. Changed STANDBY0 to STANDBY. Chapter Voltage Regulators and Power Supplies In the “Register description” section, added information on where to find the VREG_CTL base address. Revised the “Register description” section to include the address offset and MC_PCU mapping.

Table 433. Document revision history (continued)

Date	Revision	Changes
01-Oct-2011	7 (continued)	Changed WKUP to WKPU to match the official module abbreviation. In the Overview section, replaced the wakeup vector mapping information with a table. In the Overview section, changed the entries in “Interrupt vector 2” so that the footnote “Not available in 100-pin LQFP” is associated only with WKPU[19]. In the “NMI management” section, changed “This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.” to “The NIF and NOVF fields in this register are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.” In the “External interrupt management” section, changed “This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.” to “The bits in the WISR[EIF] field are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.” In the NSR, changed NIF to NIF0 and NOVF to NOVF0. In the NCR, changed all field names to contain a trailing ‘0’ (example: NLOCK0). In the “WKPU block diagram” figure, deleted single 0s. In the “Memory map” section, changed “If supported and enabled by the SoC” to “If SSCM_ERROR[RAE] is enabled”. In the WIFER section, deleted “The number of wakeups ... 1 and 18”. In the “WKPU memory map” table, added the module base address. In the NCR[NWRE0] field description, added a note about the proper sequence for enabling the NMI. Chapter Real Time Clock / Autonomous Periodic Interrupt Replaced ipg_clk with “system clock”. Changed “32 kHz” to “32 KHz”. Revised the RTCC[FRZEN] field description. Added the following note to the RTCC[RTCVAL] field description: “RTCVAL = 0 does not generate an interrupt.”. In the “RTC functional description” section, deleted “The RTCC[RTCVAL] field may only be updated when the RTCC[CNTEN] bit is cleared to disable the counter”. In the “RTC/API register map” table, added the module base address. Chapter CAN Sampler Deleted the duplicate register map. In the “CAN sampler memory map” table, added the module base address. Chapter e200z0h Core In the “e200z0h block diagram” figure, added a box around the core elements. Chapter Interrupt Controller Revised “INTC Priority Select Registers“ and “INTC Priority Select Register Address Offsets“ table to show that “INTC_PSR208_210“ contains PRI208, PRI209, and PRI210 fields. Revised the INTC_IACKR section to illustrate the register’s dependence on INTC_MCR[VTES] more clearly. In the INTC_EOIR register figure, added “See text” to the W row. In the “Interrupt vector table” table, changed “WKUP” to “WKPU”. In the “INTC memory map” table, added the module base address.

Table 433. Document revision history (continued)

Date	Revision	Changes
01-Oct-2011	7 (continued)	Chapter Memory Protection Unit In the “MPU block diagram” figure, changed the text at the top left to “Platform” and removed “z0hn1 or”. Revised the Introduction section. Changed AHB to XBAR. Deleted references to IPS and replaced with “peripheral” as needed. In the “MPU access evaluation macro” figure, changed “AHB_ap” to “System bus address phase”. In the “MPU memory map” table, added the module base address. Chapter System Integration Unit Lite Changed “WARNING” to “CAUTION”. In the register figures, changed “Access: None” to the corresponding actual level of access. In the MIDR1[PKG] field description: – Added “Any values not explicitly specified are reserved”. – Added the 64-pin LQFP setting. Revised the description of the PARTNUM field in MIDR1 and MIDR2 to clarify that the field is split between the two registers. In the PCRx section, revised the WPS and WPE field descriptions to indicate the correct functionality. In the “External interrupts” section, changed “This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.” to “The bits in the ISR[EIF] field are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.” Revised the “MIDR2 field descriptions” table to show how to calculate total flash memory size. In the “MIDR2 field descriptions” table, deleted the entry for FR (not implemented). In the “SIUL memory map” table, added the module base address. Chapter Inter-Integrated Circuit Bus Controller Module In the IBCR section, changed “MS/$\overline{SL}$” to “MSSL” and “Tx/$\overline{Rx}$” to “TXRX” to ensure compliance with field name convention. In the IBCR figure, changed bit 7 (was IBDOZE, is reserved). In the IBSR figure, changed the IBAL and IBIF fields to w1c. In the “Interrupt description” section, changed “(TCF bit set - To be checked)” to “(a Byte Transfer interrupt occurs whenever the TCF bit changes from 0 to 1, that is, Transfer Under Progress to Transfer Complete state)”. Revised the last paragraph of the Overview section. In the IBCR[MDIS] field description, added “Status register bits (IBSR) are not valid when module is disabled”. In the IBSR[RXAK] field description, added “This bit is valid only after transfer is complete”. In the “Interrupt description” section, revised the entry for “Byte transfer condition”. In the “Initialization sequence” section, changed IBCR[IBDIS] to IBCR[MDIS]. Revised the “Post-transfer software response” section. Added the “Transmit/receive sequence” section. In the “Generation of STOP” section, in the code sample, changed “bit 1” to “bit 5”. In the “I2C memory map” table, added the module base address.

Table 433. Document revision history (continued)

Date	Revision	Changes
01-Oct-2011	7 (continued)	Chapter LIN Controller In the “IFER field descriptions” table, switched “activated” and “deactivated” in order to match with “IFER[FACT] configuration” table. Deleted the “Register map and reset values” section (duplicate content). In the “UART mode” section, in the “9-bit frames” subsection, changed “sum of the 7 data bits” to “sum of the 8 data bits”. In the LINCR1[BF] field description, changed “this bit is reserved” to “this bit is reserved and always reads 1”. Changed “kbps” to “Kbit/s”. Chapter FlexCAN In the “FlexCAN memory map” table, added the module base addresses. Chapter Deserial Serial Peripheral Interface In the “Continuous selection format” section, added a note about filling the TX FIFO. Added new rules to the “Continuous serial communications clock” section. In the “DSPI memory map” table, added the module base addresses. Chapter Timers Added this chapter (incorporates content from STM, eMIOS, and PIT chapters). Chapter Analog-to-Digital Converter Updated MCR[WLSIDE] bit description. Updated CDR register. Replaced ADCDig with ADC, rewriting content as necessary. In the PDEDR[PDED] field description, added “The delay is to allow time for the ADC power supply to settle before commencing conversions.”. In the “Threshold registers” Introduction section, deleted the sentence “The inverter bit and the mask bit for mask the interrupt are stored in the TRC registers.”. Deleted the “Bit access descriptions” table. In the CIMR section, deleted the duplicate CIMR1 figure. Chapter Cross Triggering Unit Removed remaining references to CTU_CSR (not implemented on this chip). In the “CTU memory map” table: – Changed the end address of the reserved space (was 0x002C, is 0x002F). – Added the module base address. Chapter Flash Memory Replaced the entire chapter. Chapter Register Protection Added this chapter. Chapter Software Watchdog Timer Added this chapter.

Table 433. Document revision history (continued)

Date	Revision	Changes
01-Oct-2011	7 (continued)	Chapter Error Correction Status Module Revised the Introduction section. Revised the Features section. Revised the MUDCR section to show completely that bit 1 is reserved. In the register descriptions, revised the names as needed to match the names in the memory map. In the PREMR section, added text on where to find bus master IDs. Aligned register names in the descriptions and the memory map. Deleted the second paragraph in the Introduction section. Deleted the last bullet (about spp_ips_reg_protection) in the Features section. In the PREAT field descriptions, changed “AMBA-AHB” to “XBAR”. Renamed the “Spp_ips_reg_protection” section to “Register protection” and revised the section. Revised the “ECC registers” section. In the “ECSM memory map” table, added the module base address. Chapter IEEE 1149.1 Test Access Port Controller In the Features section, changed “3 test data registers” to “2 test data registers”. In the “SAMPLE instruction” section, added information about pad status. In the “SAMPLE/PRELOAD instruction” section, added information about pad status. Chapter Nexus Development Interface Added the “NPC_HNDSHK module” section. Changed HALT0 to HALT. Changed STOP0 to STOP. Changed STANDBY0 to STANDBY. Replaced the “NDI configuration options” table. Chapter Boot Assist Module Deleted this chapter (relevant content is now represented by the “Microcontroller boot” chapter). Chapter Enhanced Modular IO Subsystem Deleted this chapter (relevant content is now represented by the “Timers” chapter). Chapter System Status and Configuration Module Deleted this chapter (relevant content is now represented by the “Microcontroller Boot” chapter). Appendix: Register Protection Deleted this appendix (relevant content is now represented by the “Register Protection” chapter). Appendix: Register Map Changed HALT0 to HALT. Changed STOP0 to STOP. Changed STANDBY0 to STANDBY. Extended “Priority Select Register” to INTC_PSR208_210. Removed CTU_CSR (not implemented on this microcontroller).
17-Sep-2013	8	Updated Disclaimer