RM0070
Reference manual

SPC564Bxx, SPC56ECxx
32-bit MCU family built on the embedded Power Architecture®

Introduction

The SPC564Bxx and SPC56ECxx is a family of Power Architecture® based microcontrollers that target automotive vehicle body and gateway applications such as:

- Central body controller
- Smart junction boxes
- Front modules
- High end gateway
- Combined Body controller and gateway

The SPC564Bxx and SPC56ECxx family expands the range of the SPC560B/C microcontroller family. It provides the scalability needed to implement platform approaches and delivers the performance required by increasingly sophisticated software architectures. The advanced and cost-efficient host processor core of the SPC564Bxx and SPC56ECxx automotive controller family complies with the Power Architecture embedded category, which is 100 percent user-mode compatible with the original Power Architecture user instruction set architecture (UISA). It operates at speeds of up to 120 MHz and offers high performance processing optimized for low power consumption. It also capitalizes on the nominal 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 61

1.1 Overview 61

1.2 Audience 61

1.3 Guide to this reference manual 61

1.4 Register description conventions 65

1.5 References 66

1.6 How to use the SPC564Bxx and SPC56ECxx documents 66

1.6.1 The SPC564Bxx and SPC56ECxx document set 66

1.6.2 Reference manual content 67

1.7 Using the SPC564Bxx and SPC56ECxx 68

1.7.1 Hardware design 69

1.7.2 Input/output pins 69

1.7.3 Software design 70

1.7.4 Other features 70

2 Introduction 72

2.1 The SPC564Bxx and SPC56ECxx microcontroller family 72

2.2 SPC564Bxx and SPC56ECxx device comparison 72

2.3 Device block diagram 75

2.4 Feature summary 76

2.4.1 High-performance e200z4d core processor 76

2.4.2 e200z0h core processor 77

2.4.3 Memory Built-In Self Test (MBIST) 77

2.4.4 Enhanced Direct Memory Access Controller (eDMA) 77

2.4.5 Error Correction Status Module (ECSM) 78

2.4.6 Crossbar Switch (XBAR) 78

2.4.7 Memory Protection Unit (MPU) 78

2.4.8 Interrupt Controller (INTC) 78

2.4.9 System clocks and clock generation 79

2.4.10 System Integration Unit Lite (SIUL) 80

2.4.11 Software Watchdog Timer (SWT) 80

2.4.12 Flash memory 80

2.4.13 On-chip SRAM 83

2.4.14 Boot Assist Module (BAM) . . . 83
2.4.15 System Status and Configuration Module (SSCM) . . . 84
2.4.16 Enhanced Modular Input Output System (eMIOS) . . . 84
2.4.17 Analog-to-Digital Converter (ADC) . . . 86
2.4.18 Cross Triggering Unit (CTU) . . . 87
2.4.19 Deserial Serial Peripheral Interface (DSPI) . . . 87
2.4.20 Serial communication interface (LINFlexD) . . . 88
2.4.21 Controller Area Network (FlexCAN) . . . 89
2.4.22 Fast Ethernet Controller (FEC) . . . 90
2.4.23 Cryptographic Services Engine (CSE) . . . 90
2.4.24 Dual-Channel FlexRay Controller . . . 92
2.4.25 Inter-IC Communications (I^2C) module . . . 93
2.4.26 Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI) . . . 93
2.4.27 System Timer Module (STM) . . . 94
2.4.28 Real Time Counter/ Autonomous Periodic Interrupt (RTC/API) . . . 94
2.4.29 Nexus Development Interface (NDI) . . . 94
2.4.30 JTAG controller (JTAGC) . . . 95
2.4.31 On-chip voltage regulator (VREG) . . . 96

3 Memory Map . . . 97

4 Signal Description . . . 101

4.1 Package pinouts . . . 101
4.2 Pad configuration during reset phases . . . 105
4.3 Pad configuration during standby mode exit . . . 105
4.4 Voltage supply pins . . . 105
4.5 Pad types . . . 106
4.6 System pins . . . 106
4.7 Functional ports . . . 107
4.8 Nexus 3+ pins . . . 139

5 Microcontroller Boot . . . 140

5.1 Boot mechanism . . . 140
5.1.1 Flash memory boot . . . 141
5.1.2 Serial boot mode . . . 144
5.1.3 Censorship . . . 145

5.2 Boot Assist Module (BAM) 149
5.2.1 BAM software flow 149
5.2.2 LINFlexD (RS232) boot 157
5.2.3 FlexCAN boot 158
5.3 System Status and Configuration Module (SSCM) 159
5.3.1 Introduction 159
5.3.2 Features 160
5.3.3 Modes of operation 160
5.3.4 Memory map and register description 161

6 Clock Description 172
6.1 Clock architecture 172
6.2 Clock gating 174
6.3 Fast external crystal oscillator (FXOSC) digital interface 175
6.3.1 Main features 176
6.3.2 Functional description 176
6.3.3 Register description 177
6.4 Slow external crystal oscillator (SXOSC) digital interface 178
6.4.1 Introduction 178
6.4.2 Main features 178
6.4.3 Functional description 178
6.4.4 Register description 179
6.5 Slow internal RC oscillator (SIRC) digital interface 180
6.5.1 Introduction 180
6.5.2 Functional description 180
6.5.3 Register description 181
6.6 Fast internal RC oscillator (FIRC) digital interface 182
6.6.1 Introduction 182
6.6.2 Functional description 182
6.6.3 Register description 183
6.7 Frequency-modulated phase-locked loop (FMPLL) 184
6.7.1 Introduction 184
6.7.2 Overview 184
6.7.3 Features 184
6.7.4 Memory map 185
6.7.5 Register description 185

6.7.6 Functional description . . . 189
6.7.7 Recommendations . . . 192
6.8 Clock monitor unit (CMU) . . . 192
6.8.1 Introduction . . . 192
6.8.2 Main features . . . 192
6.8.3 Block diagram . . . 193
6.8.4 Functional description . . . 194
6.8.5 Memory map and register description . . . 195

7 Clock Generation Module (MC_CGM) . . . 200
7.1 Introduction . . . 200
7.2 Features . . . 202
7.3 External signal description . . . 202
7.4 Memory Map and Register Definition . . . 202
7.4.1 Register descriptions . . . 207
7.5 Functional description . . . 215
7.5.1 System clock generation . . . 215
7.5.2 Auxiliary clock generation . . . 217
7.5.3 Dividers functional description . . . 218
7.5.4 Output Clock Multiplexing . . . 219
7.5.5 Output Clock Division Selection . . . 219

8 Mode Entry Module (MC_ME) . . . 220
8.1 Introduction . . . 220
8.1.1 Overview . . . 220
8.1.2 Features . . . 222
8.1.3 Modes of operation . . . 222
8.2 External signal description . . . 224
8.3 Memory map and register definition . . . 224
8.3.1 Memory map . . . 224
8.3.2 Register description . . . 233
8.4 Functional description . . . 259
8.4.1 Mode transition request . . . 259
8.4.2 Mode details . . . 260
8.4.3 Mode transition process . . . 265
8.4.4 Protection of mode configuration registers . . . 273

8.4.5 Mode transition interrupts . . . 273
8.4.6 Peripheral clock gating . . . 275
8.4.7 Application example . . . 276

9 Reset Generation Module (MC_RGM) . . . 278
9.1 Introduction . . . 278
9.1.1 Overview . . . 278
9.1.2 Features . . . 280
9.1.3 Reset sources . . . 280
9.2 External signal description . . . 281
9.3 Memory map and register definition . . . 281
9.3.1 Register descriptions . . . 283
9.4 Functional description . . . 294
9.4.1 Reset state machine . . . 294
9.4.2 Destructive Resets . . . 297
9.4.3 External Reset . . . 298
9.4.4 Functional Resets . . . 298
9.4.5 STANDBY entry sequence . . . 299
9.4.6 Alternate event generation . . . 299
9.4.7 Boot mode capturing . . . 300

10 Power Control Unit (MC_PCU) . . . 301
10.1 Introduction . . . 301
10.1.1 Overview . . . 301
10.1.2 Features . . . 302
10.2 External Signal Description . . . 302
10.3 Memory Map and Register Definition . . . 303
10.3.1 Memory Map . . . 303
10.3.2 Register descriptions . . . 304
10.4 Functional description . . . 308
10.4.1 General . . . 308
10.4.2 Reset / Power-On Reset . . . 308
10.4.3 MC_PCU configuration . . . 308
10.4.4 Mode transitions . . . 309
10.5 Initialization information . . . 311
10.6 Application information . . . 311

10.6.1 STANDBY mode considerations 311

11 Voltage Regulators and Power Supplies 312

11.1 Voltage regulators 312

11.1.1 High power regulator (HPREG) 312

11.1.2 Low power regulator (LPREG) 312

11.1.3 LVDs and POR 313

11.1.4 VREG digital interface 313

11.1.5 Memory map 314

11.1.6 Register description 314

11.2 Power supply strategy 316

11.3 Power domain organization 316

12 Wakeup Unit (WKPU) 318

12.1 Overview 318

12.2 Features 321

12.3 Memory map and register description 321

12.3.1 Memory map 321

12.3.2 Register description 322

12.4 Functional description 328

12.4.1 General 328

12.4.2 Non-maskable interrupts 328

12.4.3 External wakeups/interrupts 330

12.4.4 On-chip wakeups 331

13 Real Time Clock / Autonomous Periodic Interrupt (RTC/API) 332

13.1 Overview 332

13.2 Features 332

13.3 Modes of operation 334

13.3.1 Functional mode 334

13.3.2 Debug mode 334

13.4 Register descriptions 335

13.4.1 RTC Supervisor Control Register (RTCSUPV) 335

13.4.2 RTC Control Register (RTCC) 336

13.4.3 RTC Status Register (RTCS) 338

13.4.4 RTC Counter Register (RTCCNT) 339

13.5 RTC functional description 339
13.6 API functional description 340

14 CAN Sampler 341
14.1 Introduction 341
14.2 Main features 341
14.3 Memory map and register description 342
14.3.1 Control Register (CR) 342
14.3.2 CAN Sampler Sample Registers 0–11 343
14.4 Functional description 344
14.4.1 Enabling/disabling the CAN sampler 344
14.4.2 Selecting the Rx port 345
14.4.3 Baudrate generation 345

15 e200z0h Core 346
15.1 Overview 346
15.2 Features 346
15.2.1 Microarchitecture summary 347
15.3 Core registers and programmer’s model 350
15.3.1 Unimplemented SPRs and read-only SPRs 352
15.4 Instruction summary 353

16 e200z4d Core 354
16.1 Features 356
16.1.1 Execution Unit Features 356
16.1.2 L1 Cache features 357
16.1.3 Memory management unit features 357
16.1.4 External core complex interface features 358
16.1.5 Nexus 3+ features 358
16.2 Programming model 358
16.2.1 Register set 358
16.2.2 Instruction set 361
16.2.3 Interrupts and Exception Handling 362
16.3 Microarchitecture summary 364
16.4 Availability of detailed documentation 365

17 Enhanced Direct Memory Access (eDMA) 366
17.1 Introduction 366
17.2 General features 367
17.3 Device-specific features 368
17.4 Memory map/register definition 368
17.4.1 Register descriptions 371
17.5 Functional description 391
17.5.1 eDMA Basic data flow 393
17.5.2 eDMA performance 395
17.6 Initialization / Application Information 398
17.6.1 eDMA Initialization 398
17.6.2 DMA programming errors 400
17.6.3 DMA request assignments 401
17.6.4 DMA Arbitration Mode Considerations 402
17.6.5 DMA transfer 403
17.6.6 TCD status 407
17.6.7 Channel linking 408
17.6.8 Dynamic programming 409

18 eDMA Channel Multiplexer (DMA_MUX) 412
18.1 Introduction 412
18.2 Features 412
18.2.1 Modes of operation 413
18.3 External signal description 413
18.3.1 Overview 413
18.4 Memory map and register definition 413
18.4.1 Register descriptions 414
18.4.2 DMA_MUX inputs 415
18.5 Functional description 417
18.5.1 DMA Channels with periodic triggering capability 417
18.5.2 DMA Channels with no triggering capability 419
18.5.3 "Always Enabled" DMA Sources 419
18.6 Initialization/Application Information 420
18.6.1 Reset 420
18.6.2 Enabling and Configuring Sources 420

19 Interrupt Controller (INTC) 425
19.1 Introduction 425
19.2 Features 425
19.3 Block diagram 427
19.4 Modes of operation 429
19.4.1 Software Vector mode 429
19.4.2 Hardware Vector mode 431
19.5 Memory map and register description 433
19.5.1 Memory map 433
19.5.2 Register description 433
19.6 Functional description 443
19.7 SIUL external interrupts 459
19.8 Wakeup line interrupts 459
19.9 Non-maskable interrupt (NMI) 460
19.9.1 Interrupt Request sources 460
19.9.2 Priority management 461
19.9.3 Handshaking with processor 462
19.10 Initialization/Application Information 465
19.10.1 Initialization flow 465
19.10.2 Interrupt exception handler 465
19.10.3 ISR, RTOS, and Task hierarchy 468
19.10.4 Priority Ceiling protocol 469
19.10.5 Selecting Priorities According to Request Rates and Deadlines 471
19.10.6 Software configurable Interrupt Requests 472
19.10.7 Lowering Priority Within an ISR 473
19.10.8 Negating an Interrupt Request Outside of its ISR 473
19.10.9 Examining LIFO contents 474

20 Crossbar Switch (XBAR) 475
20.1 Features 475
20.2 Introduction 476
20.2.1 Overview 476
20.2.2 Features 477
20.2.3 Limitations 477
20.2.4 General operation 477

20.3 XBAR registers 478
20.3.1 Register summary 478
20.3.2 XBAR register descriptions 479
20.3.3 Coherency 483
20.4 Function 483
20.4.1 Arbitration 483
20.4.2 Priority assignment 485
20.4.3 Master Port Functionality 485
20.4.4 Slave Port Functionality 488
20.5 Initialization/Application Information 494
20.6 Interface 494
20.6.1 Overview 494
20.6.2 Master Ports 494
20.6.3 Slave Ports 495

21 Memory protection unit (MPU) 497
21.1 Introduction 497
21.1.1 Overview 497
21.1.2 Features 498
21.1.3 Modes of operation 499
21.1.4 External signal description 499
21.2 Memory map and register description 499
21.2.1 Memory map 500
21.2.2 Register description 502
21.3 Functional description 514
21.3.1 Access evaluation macro 514
21.3.2 Putting it all together and AHB error terminations 516
21.4 Initialization information 516
21.5 Application information 517

22 Semaphores 518
22.1 Introduction 518
22.1.1 Block diagram 518
22.1.2 Features 520
22.1.3 Modes of operation 520
22.2 Signal description 520

22.3 Memory map and registers 520
22.3.1 Module memory map 520
22.3.2 Register descriptions 521
22.4 Functional description 527
22.4.1 Semaphore usage 528
22.5 Initialization information 529
22.6 Application information 529
22.7 DMA requests 530
22.8 Interrupt requests 530

23 Performance Optimization 531
23.1 Introduction 531
23.2 Features 531
23.3 Configuring hardware features 532
23.3.1 Branch target buffer (BTB) 532
23.3.2 Frequency-modulated PLL 533
23.3.3 Flash memory bus interface unit 534
23.3.4 Crossbar switch 534
23.3.5 Cache 535
23.3.6 e200z4d Memory Management Unit (MMU) 536
23.4 Application software 537
23.4.1 Compiler optimizations 537
23.4.2 e200z4d Signal Processing Extension 538
23.4.3 Hardware single precision floating point 539
23.4.4 Variable Length Encoding (VLE) 539
23.5 Peripherals and general application guidelines 540
23.6 Performance optimization checklist 540
23.6.1 Hardware configuration 540
23.6.2 Software configuration 541
23.6.3 Peripherals and general application guidelines 541

24 System Integration Unit Lite (SIUL) 542
24.1 Introduction 542
24.2 Overview 542
24.3 Features 544

24.4 External signal description 544
24.4.1 Detailed signal descriptions 544
24.5 Memory map and register description 545
24.5.1 SIUL memory map 545
24.5.2 Register protection 546
24.5.3 Register description 546
24.6 Functional description 577
24.6.1 Pad control 577
24.6.2 General purpose input and output pads (GPIO) 577
24.6.3 External interrupts 578
24.7 Pin muxing 579

25 Inter-Integrated Circuit Bus Controller Module (I^2C) 580

25.1 Introduction 580
25.1.1 Overview 580
25.1.2 Features 580
25.1.3 Block diagram 580
25.2 External signal description 581
25.2.1 SCL 581
25.2.2 SDA 581
25.3 Memory map and register description 581
25.3.1 Module memory map 581
25.3.2 I^2C Bus Address Register (IBAD) 582
25.3.3 I^2C Bus Frequency Divider Register (IBFD) 582
25.3.4 I^2C Bus Control Register (IBCR) 589
25.3.5 I^2C Bus Status Register (IBSR) 590
25.3.6 I^2C Bus Data I/O Register (IBDR) 591
25.3.7 I^2C Bus Interrupt Config Register (IBIC) 592
25.4 DMA Interface 592
25.5 Functional description 593
25.5.1 I-Bus protocol 593
25.5.2 Interrupts 597
25.6 Initialization/application information 598
25.6.1 I^2C programming examples 598
25.6.2 DMA application information 603

26 LIN Controller (LINFlexD) 608

26.1 Introduction 608

26.2 Main features 608

26.2.1 LIN mode features 609

26.2.2 UART mode features 609

26.3 The LIN protocol 610

26.3.1 Dominant and recessive logic levels 610

26.3.2 LIN frames 610

26.3.3 LIN header 611

26.3.4 Response 612

26.4 LINFlexD and software intervention 613

26.5 Summary of operating modes 613

26.6 Controller-level operating modes 614

26.6.1 Initialization mode 614

26.6.2 Normal mode 615

26.6.3 Sleep (low-power) mode 615

26.7 LIN modes 615

26.7.1 Master mode 615

26.7.2 Slave mode 617

26.7.3 Slave mode with identifier filtering 619

26.7.4 Slave mode with automatic resynchronization 622

26.8 Test modes 623

26.8.1 Loop Back mode 623

26.8.2 Self Test mode 624

26.9 UART mode 624

26.9.1 Data frame structure 624

26.9.2 Buffer 625

26.9.3 UART transmitter 626

26.9.4 UART receiver 627

26.10 Memory map and register description 629

26.10.1 LIN control register 1 (LINCR1) 631

26.10.2 LIN interrupt enable register (LINIER) 634

26.10.3 LIN status register (LINSR) 635

26.10.4 LIN error status register (LINESR) 638

26.10.5 UART mode control register (UARTCR) 639

26.10.6 UART mode status register (UARTSR) 642

26.10.7 LIN timeout control status register (LINTCSR) 644
26.10.8 LIN output compare register (LINOCR) 645
26.10.9 LIN timeout control register (LINTOCR) 645
26.10.10 LIN fractional baud rate register (LINFBRR) 646
26.10.11 LIN integer baud rate register (LINIBRR) 646
26.10.12 LIN checksum field register (LINCFR) 647
26.10.13 LIN control register 2 (LINCR2) 648
26.10.14 Buffer identifier register (BIDR) 649
26.10.15 Buffer data register least significant (BDRL) 650
26.10.16 Buffer data register most significant (BDRM) 651
26.10.17 Identifier filter enable register (IFER) 651
26.10.18 Identifier filter match index (IFMI) 652
26.10.19 Identifier filter mode register (IFMR) 653
26.10.20 Identifier filter control registers (IFCR0–IFCR15) 654
26.10.21 Global control register (GCR) 655
26.10.22 UART preset timeout register (UARTPTO) 656
26.10.23 UART current timeout register (UARTCTO) 656
26.10.24 DMA Tx enable register (DMATXE) 657
26.10.25 DMA Rx enable register (DMARXE) 658
26.11 DMA interface 658
26.11.1 Master node, TX mode 659
26.11.2 Master node, RX mode 662
26.11.3 Slave node, TX mode 664
26.11.4 Slave node, RX mode 667
26.11.5 UART node, TX mode 670
26.11.6 UART node, RX mode 672
26.11.7 Use cases and limitations 675
26.12 Functional description 676
26.12.1 8-bit timeout counter 676
26.12.2 Interrupts 677
26.12.3 Fractional baud rate generation 679
26.13 Programming considerations 680
26.13.1 Master node 681
26.13.2 Slave node 682
26.13.3 Extended frames 685
26.13.4 Timeout 686
26.13.5 UART mode 686

27 FlexCAN 687
27.1 Information specific to this device 687
27.1.1 Device-specific features 687
27.2 Introduction 687
27.2.1 Overview 688
27.2.2 FlexCAN module features 689
27.2.3 Modes of operation 690
27.3 External signal description 690
27.3.1 Overview 690
27.3.2 Signal descriptions 691
27.4 Memory map/register definition 691
27.4.1 FlexCAN memory mapping 691
27.4.2 Message Buffer Structure 693
27.4.3 Rx FIFO structure 696
27.4.4 Register descriptions 698
27.5 Functional description 716
27.5.1 Overview 716
27.5.2 Local Priority Transmission 717
27.5.3 Transmit process 717
27.5.4 Arbitration process 717
27.5.5 Receive process 718
27.5.6 Data coherence 720
27.5.7 Rx FIFO 721
27.5.8 CAN Protocol Related Features 722
27.5.9 Modes of operation details 726
27.5.10 Interrupts 727
27.5.11 Bus interface 727
27.6 Initialization/application information 728
27.6.1 FlexCAN initialization sequence 728
27.6.2 FlexCAN Addressing and RAM size configurations 729

28 Deserial Serial Peripheral Interface (DSPI) 730
28.1 Introduction 730
28.1.1 Features 731
28.1.2 DSPI configurations 732
28.1.3 Modes of operation 733

28.2 External signal description 734
28.2.1 Overview 734
28.2.2 Detailed signal description 734
28.3 Memory map and register definition 735
28.3.1 Memory map 735
28.3.2 Register descriptions 737
28.4 Functional Description 759
28.4.1 Start and Stop of DSPI Transfers 760
28.4.2 Serial Peripheral Interface (SPI) Configuration 761
28.4.3 Deserial Serial Interface (DSI) Configuration 763
28.4.4 Combined Serial Interface (CSI) Configuration 768
28.4.5 DSPI Baud Rate and Clock Delay Generation 769
28.4.6 Transfer Formats 772
28.4.7 Continuous Serial Communications Clock 781
28.4.8 Slave Mode Operation Constraints 782
28.4.9 Timed Serial Bus (TSB) 783
28.4.10 Parity Generation and Check. 786
28.4.11 Interrupts/DMA Requests 787
28.4.12 Power Saving Features 788
28.5 Initialization/Application Information 789
28.5.1 How to Manage DSPI Queues 789
28.5.2 Switching Master and Slave Mode 790
28.5.3 Baud Rate Settings 790
28.5.4 Delay Settings 791
28.5.5 Calculation of FIFO Pointer Addresses 792

29 FlexRay Communication Controller (FLEXRAY) 795

29.1 Introduction 795
29.1.1 Reference 795
29.1.2 Glossary 795
29.1.3 Color Coding 796
29.1.4 Overview 796
29.1.5 Features 798
29.1.6 Modes of Operation 799
29.2 External Signal Description 800
29.2.1 Detailed Signal Descriptions 800

29.3 Controller Host Interface Clocking 801
29.4 Protocol Engine Clocking 801
29.4.1 Oscillator Clocking 801
29.4.2 PLL Clocking 801
29.5 Memory Map and Register Description 801
29.5.1 Memory Map 802
29.5.2 Register Descriptions 805
29.6 Functional Description 884
29.6.1 Message Buffer Concept 884
29.6.2 Physical Message Buffer 884
29.6.3 Message Buffer Types 885
29.6.4 Flexray Memory Area Layout 891
29.6.5 Physical Message Buffer Description 894
29.6.6 Individual Message Buffer Functional Description 903
29.6.7 Individual Message Buffer Search 919
29.6.8 Individual Message Buffer Reconfiguration 922
29.6.9 Receive FIFOs 923
29.6.10 Channel Device Modes 929
29.6.11 External Clock Synchronization 931
29.6.12 Sync Frame ID and Sync Frame Deviation Tables 932
29.6.13 MTS Generation 935
29.6.14 Key Slot Transmission 936
29.6.15 Sync Frame Filtering 936
29.6.16 Strobe Signal Support 937
29.6.17 Timer Support 938
29.6.18 Slot Status Monitoring 939
29.6.19 System Bus Access 943
29.6.20 Interrupt Support 944
29.6.21 Lower Bit Rate Support 948
29.6.22 PE Data Memory (PE DRAM) 949
29.6.23 CHI Lookup-Table Memory (CHI LRAM) 950
29.6.24 Memory Content Error Detection 951
29.6.25 Memory Error Injection 955
29.7 Application Information 957
29.7.1 Module Configuration 957
29.7.2 Initialization Sequence 958

29.7.3 Memory Error Injection out of POC:default config . . . 959
29.7.4 Shut Down Sequence . . . 960
29.7.5 Number of Usable Message Buffers . . . 960
29.7.6 Protocol Control Command Execution . . . 961
29.7.7 Message Buffer Search on Simple Message Buffer Configuration . . . 962

30 Fast Ethernet Controller (FEC) . . . 965
30.1 Overview . . . 965
30.1.1 Features . . . 965
30.2 Modes of Operation . . . 965
30.2.1 Full and Half Duplex Operation . . . 966
30.2.2 Interface Options . . . 966
30.2.3 Address Recognition Options . . . 966
30.2.4 Internal Loopback . . . 966
30.3 FEC Top-Level Functional Diagram . . . 966
30.4 Functional Description . . . 968
30.4.1 Initialization Sequence . . . 968
30.4.2 User Initialization (Prior to Asserting ECR[ETHER_EN]) . . . 969
30.4.3 Microcontroller Initialization . . . 970
30.4.4 User Initialization (After Asserting ECR[ETHER_EN]) . . . 970
30.4.5 Network Interface Options . . . 970
30.4.6 FEC Frame Transmission . . . 971
30.4.7 FEC Frame Reception . . . 972
30.4.8 Ethernet Address Recognition . . . 973
30.4.9 Hash Algorithm . . . 975
30.4.10 Full Duplex Flow Control . . . 978
30.4.11 Inter-Packet Gap (IPG) Time . . . 978
30.4.12 Collision Handling . . . 979
30.4.13 Internal and External Loopback . . . 979
30.4.14 Ethernet Error-Handling Procedure . . . 979
30.5 Programming Model . . . 981
30.5.1 Top Level Module Memory Map . . . 981
30.5.2 Register map . . . 981
30.5.3 MIB Block Counters Memory Map . . . 982
30.5.4 Registers . . . 984
30.6 Buffer Descriptors . . . 1000

30.6.1 Driver/DMA Operation with Buffer Descriptors 1000
30.6.2 Ethernet Receive Buffer Descriptor (RxBD) 1002
30.6.3 Ethernet Transmit Buffer Descriptor (TxBD) 1003

31 Timers 1006
31.1 Technical overview 1006
31.1.1 Overview of the STM 1008
31.1.2 Overview of the eMIOS 1008
31.1.3 Overview of the PIT_RTI 1010
31.2 System Timer Module (STM) 1010
31.2.1 Introduction 1010
31.2.2 External signal description 1011
31.2.3 Memory map and register definition 1011
31.2.4 Functional description 1014
31.3 Enhanced Modular IO Subsystem (eMIOS) 1015
31.3.1 Introduction 1015
31.3.2 External signal description 1018
31.3.3 Memory map and register description 1018
31.3.4 Functional description 1029
31.3.5 Initialization/Application information 1059
31.4 Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI) 1063
31.4.1 Introduction 1063
31.4.2 Features 1064
31.4.3 Modes of operation 1064
31.4.4 Signal description 1064
31.4.5 Memory map and register description 1064
31.4.6 Functional description 1068
31.4.7 Initialization and application information 1070

32 Analog-to-Digital Converter (ADC) 1071
32.1 Overview 1071
32.1.1 Device-specific pin configuration features 1071
32.1.2 Device-specific implementation 1072
32.2 Introduction 1075
32.3 Register descriptions 1076
32.3.1 Introduction 1076

32.3.2 Control logic registers 1083
32.3.3 Interrupt registers 1086
32.3.4 DMA registers 1094
32.3.5 Threshold Register 1097
32.3.6 Presampling registers 1099
32.3.7 Mask registers 1102
32.3.8 Delay registers 1106
32.3.9 Data registers 1107
32.4 Functional description 1120
32.4.1 Analog channel conversion 1120
32.4.2 Analog clock generator and conversion timings 1123
32.4.3 ADC sampling and conversion timing 1124
32.4.4 ADC CTU (Cross Triggering Unit) 1126
32.4.5 Presampling 1126
32.4.6 Programmable analog watchdog 1127
32.4.7 DMA functionality 1129
32.4.8 Interrupts 1129
32.4.9 External decode signals delay 1129
32.4.10 Power-down mode 1130
32.4.11 Auto-clock-off mode 1130

33 Cross Triggering Unit (CTU) 1131
33.1 Introduction 1131
33.2 Main features 1131
33.3 Block diagram 1131
33.4 Memory map and register descriptions 1132
33.4.1 Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63) 1132
33.5 Functional description 1133
33.5.1 Channel value 1135

34 Static RAM (SRAM) 1138
34.1 Introduction 1138
34.2 SRAM operating mode 1138
34.3 Register memory map 1138
34.4 SRAM ECC mechanism 1139
34.4.1 Access timing 1139

34.4.2 Reset effects on SRAM accesses 1140
34.5 Functional description 1140
34.6 Initialization and application information 1141

35 Flash Memory 1142
35.1 Introduction 1142
35.2 Code flash memory 1142
35.2.1 Introduction 1142
35.2.2 Main features 1143
35.2.3 Block diagram 1143
35.2.4 Functional description 1144
35.2.5 Register description 1150
35.2.6 STCU programming using Flash 1183
35.2.7 Programming considerations 1185
35.3 Data flash memory 1195
35.3.1 Introduction 1195
35.3.2 Main features 1196
35.3.3 Block diagram 1196
35.3.4 Functional description 1197
35.3.5 User mode operation 1199
35.3.6 Register description 1200
35.3.7 Programming considerations 1216
35.3.8 Error correction code 1222
35.3.9 Protection strategy 1223
35.4 Platform Flash Controller 1224
35.4.1 Introduction 1224
35.4.2 External Signal Descriptions 1226
35.4.3 Memory map and register description 1227
35.4.4 Functional Description 1238

36 Register Protection 1254
36.1 Introduction 1254
36.1.1 Overview 1254
36.1.2 Features 1255
36.1.3 Modes of operation 1255
36.2 External signal description 1255

36.3 Memory map and register description 1255
36.3.1 Memory map 1257
36.3.2 Register description 1257
36.4 Functional description 1260
36.4.1 General 1260
36.4.2 Change lock settings 1260
36.4.3 Access errors 1263
36.5 Reset 1264
36.6 Protected registers 1264

37 Software Watchdog Timer (SWT) 1276
37.1 Introduction 1276
37.2 Features 1276
37.3 Modes of operation 1276
37.4 External signal description 1276
37.5 Memory map and register definition 1276
37.5.1 Memory map 1277
37.5.2 Register descriptions 1277
37.6 Functional Description 1281

38 Error Correction Status Module (ECSM) 1284
38.1 Introduction 1284
38.2 Overview 1284
38.3 Features 1284
38.4 Memory map and register description 1284
38.4.1 Memory map 1284
38.4.2 Register description 1285
38.4.3 Register protection 1302

39 Self-Test Control Unit (STCU) 1303
39.1 Introduction 1303
39.1.1 Acronyms, abbreviations, and terms 1303
39.2 STCU main features 1303
39.3 Block diagram and components 1304
39.4 The Safety Integrity Subsystem 1305

39.4.1 Default setup after the boot sequence phase 1 1307
39.4.2 Changing the default fault grading 1311
39.4.3 Integrity SW operations 1312
39.5 Memory map and register definition 1312
39.5.1 Memory map 1312
39.5.2 Register conventions 1313
39.5.3 Detailed register descriptions 1314
39.5.4 Self-Test sequence after reset trigger 1330

40 Cryptographic Services Engine (CSE) 1332
40.1 Introduction 1332
40.1.1 Overview 1332
40.1.2 Features 1332
40.1.3 Modes of operation 1332
40.1.4 Block diagram 1332
40.2 External signal description 1333
40.3 Memory map and register definition 1333
40.3.1 Memory map 1333
40.3.2 Register descriptions 1334
40.4 CSE functional description 1340
40.4.1 Host Interface 1340
40.4.2 Command Processing 1340
40.4.3 Secure Storage 1341
40.4.4 Encryption and Decryption 1343
40.4.5 Message Authentication 1343
40.4.6 Secure Boot 1343
40.4.7 Random Number Generation 1343
40.4.8 Error Handling 1344
40.5 CSE Commands 1345
40.5.1 Encrypt ECB 1346
40.5.2 Encrypt CBC 1346
40.5.3 Decrypt ECB 1346
40.5.4 Decrypt CBC 1347
40.5.5 Generate MAC 1347
40.5.6 Verify MAC 1348
40.5.7 Load Key 1348

40.5.8 Load Plain Key . . . 1349
40.5.9 Export RAM Key . . . 1349
40.5.10 Initialize RNG . . . 1350
40.5.11 Extend PRNG Seed . . . 1350
40.5.12 Generate Random Number . . . 1350
40.5.13 Secure Boot . . . 1351
40.5.14 Boot Failure . . . 1351
40.5.15 Boot OK . . . 1351
40.5.16 Get ID . . . 1352
40.5.17 Cancel . . . 1352
40.5.18 Debug Challenge . . . 1352
40.5.19 Debug Authorization . . . 1353
40.5.20 Generate TRNG Random Number . . . 1353
40.5.21 Initialize CSE . . . 1354

41 JTAG Controller (JTAGC) . . . 1355
41.1 Introduction . . . 1355
41.1.1 Overview . . . 1355
41.1.2 Features . . . 1356
41.1.3 Modes of Operation . . . 1356
41.2 External signal description . . . 1357
41.2.1 Overview . . . 1357
41.2.2 Detailed signal descriptions . . . 1357
41.3 Register definition . . . 1357
41.3.1 Register Descriptions . . . 1358
41.4 Functional Description . . . 1360
41.4.1 JTAGC Reset Configuration . . . 1360
41.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port . . . 1360
41.4.3 TAP Controller State Machine . . . 1360
41.4.4 JTAGC Block Instructions . . . 1362
41.4.5 Boundary Scan . . . 1364
41.5 Initialization/Application Information . . . 1364

42 Nexus Development Interface (NDI) . . . 1365
42.1 Introduction . . . 1365
42.2 Block diagram . . . 1365

42.2.1 NDI Features 1368
42.2.2 Modes of Operation 1369
42.3 External Signal Description 1370
42.4 Memory Map and Registers 1371
42.4.1 NDI Functional Description 1373
42.5 Nexus Port Controller (NPC) 1376
42.5.1 Introduction 1376
42.5.2 NPC features 1376
42.5.3 NPC memory map 1377
42.5.4 NPC Register descriptions 1377
42.5.5 NPC Functional Description 1380
42.5.6 NPC Initialization/Application Information 1387
42.6 Nexus3+ Module 1388
42.6.1 Introduction 1388
42.6.2 Block Diagram 1388
42.6.3 Overview 1389
42.6.4 Enabling Nexus3 Operation 1389
42.6.5 TCODEs Supported 1390
42.6.6 Memory Map 1395
42.6.7 Register Definition 1396
42.6.8 Register Access via JTAG / OnCE 1415
42.7 Debug Implementation 1416
42.8 Debug Capabilities 1416
42.9 Debug Port 1417
42.9.1 Nexus3+ Auxiliary Port 1418

43 Revision history 1419

List of tables

Table 1. Guide to this reference manual . . . 61
Table 2. Reference manual integration and functional content . . . 68
Table 3. SPC564Bxx and SPC56ECxx family comparison . . . 73
Table 4. SPC564Bxx and SPC56ECxx flash memory partitioning . . . 81
Table 5. Supported eMIOS channel modes . . . 84
Table 6. eMIOS configuration . . . 85
Table 7. SPC564Bxx and SPC56ECxx memory map . . . 97
Table 8. Voltage supply pin descriptions . . . 105
Table 9. System pin descriptions . . . 107
Table 10. Functional port pin descriptions . . . 108
Table 11. Nexus 3+ pin descriptions . . . 139
Table 12. Boot mode selection . . . 140
Table 13. RCHW field descriptions . . . 142
Table 14. Examples of legal and illegal passwords . . . 145
Table 15. Censorship configuration and truth table . . . 146
Table 16. SSCM_STATUS[BMODE] values as used by BAM . . . 151
Table 17. Serial boot mode – baud rates . . . 151
Table 18. BAM censorship mode detection . . . 152
Table 19. UART boot mode download protocol . . . 157
Table 20. FlexCAN boot mode download protocol . . . 159
Table 21. SSCM memory map . . . 161
Table 22. SSCM_STATUS allowed register accesses . . . 162
Table 23. SSCM_STATUS field descriptions . . . 162
Table 24. SSCM_MEMCONFIG field descriptions . . . 163
Table 25. SSCM_MEMCONFIG allowed register accesses . . . 164
Table 26. SSCM_ERROR field descriptions . . . 164
Table 27. SSCM_ERROR allowed register accesses . . . 165
Table 28. SSCM_DEBUGPORT field descriptions . . . 165
Table 29. Debug status port modes . . . 165
Table 30. SSCM_DEBUGPORT allowed register accesses . . . 166
Table 31. Password Comparison Register field descriptions . . . 167
Table 32. SSCM_PWCMPH/L allowed register accesses . . . 167
Table 33. SSCM_DPMBOOT field descriptions . . . 168
Table 34. SSCM_DPMKEY field descriptions . . . 169
Table 35. SSCM_UOPS field descriptions . . . 169
Table 36. SSCM_PSA field descriptions . . . 170
Table 37. SSCM_CLEN field descriptions . . . 171
Table 38. SPC564Bxx and SPC56ECxx — Peripheral clock sources . . . 174
Table 39. SPC564Bxx and SPC56ECxx example peripheral clock divider setup . . . 175
Table 40. Truth table of crystal oscillator . . . 176
Table 41. FXOSC_CTL field descriptions . . . 177
Table 42. SXOSC truth table . . . 178
Table 43. SXOSC_CTL field descriptions . . . 179
Table 44. SIRC_CTL field descriptions . . . 181
Table 45. FIRC_CTL field descriptions . . . 183
Table 46. FMPLL memory map . . . 185
Table 47. CR field descriptions . . . 186
Table 48. Input divide ratios . . . 187

Table 49. Output divide ratios . . . 187
Table 50. Loop divide ratios . . . 188
Table 51. MR field descriptions . . . 189
Table 52. FMPLL lookup table . . . 190
Table 53. Progressive clock switching on pll_select rising edge . . . 190
Table 54. CMU memory map . . . 195
Table 55. CMU_CSR field descriptions . . . 196
Table 56. CMU_FDR field descriptions . . . 197
Table 57. CMU_HFREFR field descriptions . . . 197
Table 58. CMU_LFREFR field descriptions . . . 198
Table 59. CMU_ISR field descriptions . . . 199
Table 60. CMU_MDR field descriptions . . . 199
Table 61. MC_CGM Register Description . . . 202
Table 62. MC_CGM Memory Map . . . 203
Table 63. e200z0h Core Clock Divider Configuration Register (CGM_Z0_DCR) Field Description . 208
Table 64. FEC Clock Divider Configuration Register (CGM_FEC_DCR) Field Description . . . 209
Table 65. Flash Clock Divider Configuration Register (CGM_Flash_DCR) Field Description . . . 209
Table 66. Output Clock Enable Register (CGM_OC_EN) Field Descriptions . . . 210
Table 67. Output Clock Division Select Register (CGM_OCDS_SC) Field Descriptions . . . 210
Table 68. System Clock Select Status Register (CGM_SC_SS) Field Descriptions . . . 211
Table 69. System Clock Divider 0 Configuration Register (CGM_SC_DC0) Field Descriptions . . . 212
Table 70. System Clock Divider 1 Configuration Register (CGM_SC_DC1) Field Descriptions . . . 212
Table 71. System Clock Divider 2 Configuration Register (CGM_SC_DC2) Field Descriptions . . . 213
Table 72. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) Field Descriptions . . . 213
Table 73. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) Field Descriptions . . . 214
Table 74. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0) Field Descriptions 215
Table 75. MC_ME mode descriptions . . . 223
Table 76. MC_ME register description . . . 224
Table 77. MC_ME memory map . . . 228
Table 78. Global Status Register (ME_GS) Field Descriptions . . . 234
Table 79. Mode Control Register (ME_MCTL) Field Descriptions . . . 236
Table 80. Mode Enable Register (ME_ME) Field Descriptions . . . 237
Table 81. Interrupt Status Register (ME_IS) Field Descriptions . . . 239
Table 82. Interrupt Mask Register (ME_IM) Field Descriptions . . . 240
Table 83. Invalid Mode Transition Status Register (ME_IMTS) Field Descriptions . . . 241
Table 84. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions . . . 243
Table 85. Mode Configuration Registers (ME_<mode>_MC) field descriptions . . . 250
Table 86. Peripheral Status Registers (ME_PSn) Field Descriptions . . . 254
Table 87. Run Peripheral Configuration Registers (ME_RUN_PC0...7) Field Descriptions . . . 255
Table 88. Low-Power Peripheral Configuration Registers (ME_LP_PC0...7) Field Descriptions . . . 256
Table 89. Peripheral Control Registers (ME_PCTLn) Field Descriptions . . . 257
Table 90. Peripheral control registers by peripheral . . . 257
Table 91. MC_ME resource control overview . . . 266
Table 92. MC_ME system clock selection overview . . . 269
Table 93. MC_RGM register description . . . 281
Table 94. MC_RGM memory map . . . 282
Table 95. Functional Event Status Register (RGM_FES) Field Descriptions . . . 284
Table 96. Destructive Event Status Register (RGM_DES) field descriptions . . . 286
Table 97. Functional Event Reset Disable Register (RGM_FERD) field descriptions . . . 287
Table 98. Destructive Event Reset Disable Register (RGM_DERD) field descriptions . . . 289
Table 99. Functional Event Alternate Request Register (RGM_FEAR) field descriptions . . . 289
Table 100. Functional Event Short Sequence Register (RGM_FESS) field descriptions . . . 291

Table 101. STANDBY Reset Sequence Register (RGM_STDBY) field descriptions 293
Table 102. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions. 293
Table 103. MC_RGM Reset implications 294
Table 104. MC_RGM alternate event selection 299
Table 105. MC_PCU Register Description 303
Table 106. MC_PCU Memory Map. 303
Table 107. Power Domain Configuration register field descriptions 305
Table 108. RAM configurations in modes. 307
Table 109. PCONF settings for RAM in STANDBY 307
Table 110. Power Domain Status Register (PCU_PSTAT) Field Descriptions. 308
Table 111. Memory map. 314
Table 112. VREG_CTL field descriptions 315
Table 113. VREG_PDMODE field descriptions 316
Table 114. Wakeup vector mapping 318
Table 115. WKPU memory map 321
Table 116. NSR field descriptions 323
Table 117. NCR field descriptions 324
Table 118. WISR field descriptions. 325
Table 119. IRER field descriptions 326
Table 120. WRER field descriptions 326
Table 121. WIREER field descriptions 327
Table 122. WIFEER field descriptions 327
Table 123. WIFER field descriptions 327
Table 124. WIPUER field descriptions 328
Table 125. RTC/API register map. 335
Table 126. RTCSUPV register field descriptions 335
Table 127. RTCC register field descriptions 336
Table 128. RTCS register field descriptions 338
Table 129. RTCCNT register field descriptions 339
Table 130. CAN Sampler memory map 342
Table 131. CR field descriptions. 342
Table 132. Internal multiplexer correspondence. 345
Table 133. List of Unimplemented SPRs 353
Table 134. Interrupt Registers 362
Table 135. Exceptions and conditions 363
Table 136. eDMA memory map 369
Table 137. eDMA 32-bit memory map 370
Table 138. DMA Control Register (EDMA_CR) field descriptions 372
Table 139. DMA Error Status (EDMA_ESR) field descriptions. 374
Table 140. DMA Enable Request (EDMA_ERQRL) field descriptions 376
Table 141. DMA Enable Error Interrupt (EDMA_EEIRL) field descriptions 377
Table 142. DMA Set Enable Request (EDMA_SERQR) field descriptions. 377
Table 143. DMA Clear Enable Request (EDMA_CERQR) field descriptions 378
Table 144. DMA Set Enable Error Interrupt (EDMA_SEEIR) field descriptions 378
Table 145. DMA Clear Enable Error Interrupt (EDMA_CEEIR) field descriptions 379
Table 146. DMA Clear Interrupt Request (EDMA_CIRQR) field descriptions 379
Table 147. DMA Clear Error (EDMA_CER) field descriptions 380
Table 148. DMA Set START Bit (EDMA_SSBR) field descriptions 380
Table 149. DMA Clear DONE Status (EDMA_CDSBR) field descriptions 381
Table 150. DMA Interrupt Request (EDMA_IRQRL) field descriptions. 382
Table 151. DMA Error (EDMA_ERL) field descriptions 382
Table 152. DMA Hardware Request Status (EDMA_HRSL) field descriptions. 383

Table 153. DMA Channel n Priority (EDMA_CPRn) field descriptions . . . 384
Table 154. TCDn 32-bit Memory Structure . . . 384
Table 155. TCDn field descriptions . . . 386
Table 156. eDMA peak transfer rates [MB/s] . . . 396
Table 157. eDMA peak request rate [MReq/sec] . . . 397
Table 158. TCD Primary Control and Status Fields . . . 399
Table 159. DMA Request Summary for eDMA . . . 401
Table 160. Modulo Feature Example . . . 406
Table 161. Channel Linking Parameters . . . 409
Table 162. Coherency model for a dynamic channel link request . . . 409
Table 163. Coherency model for method 1 . . . 410
Table 164. Coherency model for method 2 . . . 411
Table 165. DMA_MUX memory map . . . 413
Table 166. CHCONFIGxx Field Descriptions . . . 414
Table 167. Channel and Trigger Enabling . . . 414
Table 168. DMA channel mapping . . . 415
Table 169. DMA_MUX periodic trigger inputs . . . 417
Table 170. Interrupt sources available . . . 426
Table 171. INTC memory map . . . 433
Table 172. INTC_MCR Field Descriptions . . . 434
Table 173. INTC_CPR_PRC0 Field Descriptions . . . 435
Table 174. PRI values . . . 435
Table 175. INTC_IACKR_PRC0 Field Descriptions . . . 437
Table 176. INTC_IACKR_PRC1 Field Descriptions . . . 438
Table 177. INTC_SSCIR[0:7] Field Descriptions . . . 440
Table 178. INTC_PSR0_3–INTC_PSR275-278 Field Descriptions . . . 441
Table 179. INTC Priority Select Register Address Offsets . . . 441
Table 180. Selected Processor for Interrupt Request . . . 442
Table 181. e200z0h core interrupts . . . 443
Table 182. e200z4d core interrupts . . . 444
Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table . . . 445
Table 184. Order of ISR Execution Example . . . 468
Table 185. Raised Priority Preserved Events . . . 471
Table 186. Master/slave mappings . . . 475
Table 187. XBAR memory map . . . 478
Table 188. MPRn field descriptions . . . 479
Table 189. SGPCRn field descriptions . . . 481
Table 190. MGPCRn field descriptions . . . 482
Table 191. MPU memory map . . . 500
Table 192. MPU_CESR field descriptions . . . 503
Table 193. MPU_EARn field descriptions . . . 504
Table 194. MPU_EDRn field descriptions . . . 505
Table 195. MPU_RGDn.Word0 field descriptions . . . 506
Table 196. MPU_RGDn.Word1 field descriptions . . . 507
Table 197. MPU_RGDn.Word2 field descriptions . . . 508
Table 198. MPU_RGDn.Word3 field descriptions . . . 511
Table 199. MPU_RGDAACn field descriptions . . . 512
Table 200. Protection violation definition . . . 515
Table 201. Semaphores memory map . . . 520
Table 202. SEMA4_GATE*n* Field Descriptions . . . 522
Table 203. SEMA4_CP{0,1}NTF Field Descriptions . . . 523
Table 204. SEMA4_CP{0,1}NTF Field Descriptions . . . 524

Table 205. SEMA4_RSTGT Field Descriptions . . . 525
Table 206. SEMA4_RSTGT Field Descriptions . . . 526
Table 207. BUCSR field descriptions . . . 533
Table 208. L1CSR1 field descriptions . . . 535
Table 209. MAS2 field descriptions . . . 537
Table 210. MSR field descriptions . . . 539
Table 211. Performance optimization — hardware configuration . . . 540
Table 212. Performance optimization — software configuration . . . 541
Table 213. SIUL signal properties . . . 544
Table 214. SIUL memory map . . . 545
Table 215. MIDR1 field descriptions . . . 547
Table 216. MIDR2 field descriptions . . . 548
Table 217. ISR field descriptions . . . 549
Table 218. IRER field descriptions . . . 549
Table 219. IREER field descriptions . . . 550
Table 220. IFEER field descriptions . . . 551
Table 221. IFER field descriptions . . . 551
Table 222. PCR bit implementation by pad type . . . 552
Table 223. PCRx field descriptions . . . 553
Table 224. PSMI field descriptions . . . 554
Table 225. Peripheral input pin selection . . . 555
Table 226. GPDO0_3 field descriptions . . . 560
Table 227. GPDO0_3 field descriptions . . . 561
Table 228. PGPDO0 – PGPDO6 Register Map . . . 561
Table 229. PGPDI0 – PGPDI6 Register Map . . . 562
Table 230. MPGPDO0 – MPGPDO12 Register Map . . . 563
Table 231. MPGPDO0..MPGPDO12 field descriptions . . . 564
Table 232. IFMC field descriptions . . . 564
Table 233. IFCPR field descriptions . . . 565
Table 234. eMIOS channel inputs for each output channel . . . 573
Table 235. DISR field descriptions . . . 575
Table 236. I2C memory map . . . 582
Table 237. IBAD field descriptions . . . 582
Table 238. IBFD field descriptions . . . 583
Table 239. I-Bus multiplier factor . . . 583
Table 240. I-Bus prescaler divider values . . . 583
Table 241. I-Bus tap and prescale values . . . 583
Table 242. I^2C divider and hold values . . . 586
Table 243. IBCR field descriptions . . . 589
Table 244. IBSR Field Descriptions . . . 590
Table 245. IBIC field descriptions . . . 592
Table 246. Interrupt summary . . . 597
Table 247. Errors in Master mode . . . 616
Table 248. Errors in Slave mode . . . 618
Table 249. Filter submodes . . . 620
Table 250. Filter to interrupt vector correlation . . . 621
Table 251. UART buffer structure . . . 626
Table 252. BDRL access in UART mode . . . 626
Table 253. BDRM access in UART mode . . . 627
Table 254. UART receiver scenarios . . . 628
Table 255. LINFlexD_0 memory map . . . 629
Table 256. LINFlexD_1–9 memory map . . . 630

Table 257. LINCR1 field descriptions . . . 631
Table 258. Checksum bits configuration . . . 632
Table 259. LIN master break length selection . . . 633
Table 260. Operating mode selection . . . 633
Table 261. LINIER field descriptions . . . 634
Table 262. LINSR field descriptions . . . 636
Table 263. LINESR field descriptions . . . 638
Table 264. UARTCR field descriptions . . . 640
Table 265. UARTSR field descriptions . . . 642
Table 266. LINTCSR field descriptions . . . 644
Table 267. LINOCR field descriptions . . . 645
Table 268. LINTOCR field descriptions . . . 646
Table 269. LINFBRR field descriptions . . . 646
Table 270. LINIBRR field descriptions . . . 647
Table 271. Integer baud rate selection . . . 647
Table 272. LINCFR field descriptions . . . 647
Table 273. LINCR2 field descriptions . . . 648
Table 274. BIDR field descriptions . . . 649
Table 275. BDRL field descriptions . . . 650
Table 276. BDRM field descriptions . . . 651
Table 277. IFER field descriptions . . . 652
Table 278. IFMI field descriptions . . . 652
Table 279. IFMR field descriptions . . . 653
Table 280. IFMR[IFM] configuration . . . 653
Table 281. IFCR functionality based on mode . . . 654
Table 282. IFCR field descriptions . . . 654
Table 283. GCR field descriptions . . . 655
Table 284. UARTPTO field descriptions . . . 656
Table 285. UARTCTO field descriptions . . . 657
Table 286. DMATXE field descriptions . . . 658
Table 287. DMARXE field descriptions . . . 658
Table 288. Register settings (master node, TX mode) . . . 660
Table 289. TCD settings (master node, TX mode) . . . 662
Table 290. TCD settings (master node, RX mode) . . . 664
Table 291. Register settings (slave node, TX mode) . . . 665
Table 292. TCD settings (slave node, TX mode) . . . 667
Table 293. Register settings (slave node, RX mode) . . . 668
Table 294. TCD settings (slave node, RX mode) . . . 669
Table 295. TCD settings (UART node, TX mode) . . . 672
Table 296. TCD settings (UART node, RX mode) . . . 675
Table 297. LINFlexD interrupt control . . . 677
Table 298. Error calculation for programmed baud rates . . . 680
Table 299. FlexCAN Signals . . . 691
Table 300. FlexCAN memory map . . . 692
Table 301. Message Buffer MB0 memory mapping . . . 693
Table 302. Message Buffer Structure field description . . . 693
Table 303. Message Buffer Code for Rx buffers . . . 695
Table 304. Message Buffer Code for Tx buffers . . . 696
Table 305. Rx FIFO Structure field description . . . 698
Table 306. MCR field descriptions . . . 699
Table 307. IDAM coding . . . 702
Table 308. CTRL field descriptions . . . 703

Table 309. TIMER field descriptions . . . 706
Table 310. RXGMASK field descriptions . . . 708
Table 311. ECR field descriptions . . . 710
Table 312. ESR field descriptions . . . 710
Table 313. Fault confinement state . . . 712
Table 314. MASK2 field descriptions . . . 713
Table 315. IMASK1 field descriptions . . . 714
Table 316. IFLAG2 field descriptions . . . 714
Table 317. IFLAG1 field descriptions . . . 715
Table 318. RXIMR0–RXIMR63 field descriptions . . . 716
Table 319. Time Segment Syntax . . . 724
Table 320. Bosch CAN 2.0B standard compliant bit time segment settings . . . 724
Table 321. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate . . . 725
Table 322. Signal properties . . . 734
Table 323. DSPI configuration . . . 734
Table 324. DSPI memory map . . . 736
Table 325. DSPI_MCR Field Descriptions . . . 738
Table 326. DSPI_TCR Field Descriptions . . . 740
Table 327. DSPI_CTAR*n* Field Descriptions in master mode . . . 742
Table 328. DSPI SCK Duty Cycle . . . 744
Table 329. Delay Scaler Encoding . . . 744
Table 330. DSPI Baud Rate Scaler . . . 745
Table 331. DSPI_CTAR*0,* DSPI_CTAR*1* Field Descriptions in slave mode . . . 745
Table 332. DSPI_SR Field Descriptions . . . 746
Table 333. DSPI_RSER Field Descriptions . . . 748
Table 334. DSPI_PUSHR Field Descriptions in master mode . . . 750
Table 335. DSPI_PUSHR Field Descriptions in slave mode . . . 751
Table 336. DSPI_POPR Field Descriptions . . . 751
Table 337. DSPI_TXFR*n* Field Descriptions . . . 752
Table 338. DSPI_RXFR*n* Field Descriptions . . . 753
Table 339. DSPI_DSICR Field Descriptions . . . 753
Table 340. DSPI_SDR Field Descriptions . . . 755
Table 341. DSPI_ASDR Field Descriptions . . . 755
Table 342. DSPI_COMPR Field Descriptions . . . 756
Table 343. DSPI_DDR Field Descriptions . . . 756
Table 344. DSPI_DSICR1 Field Descriptions . . . 757
Table 345. DSPI_SSR Field Descriptions . . . 758
Table 346. DSPI_DIMR Field Descriptions . . . 759
Table 347. DSPI_DPIR Field Descriptions . . . 759
Table 348. DSI Data Transfer Initiation Control . . . 766
Table 349. Baud Rate Computation Example . . . 770
Table 350. CS to SCK delay computation example . . . 770
Table 351. After SCK Delay Computation Example . . . 770
Table 352. Delay after Transfer Computation Example . . . 770
Table 353. Peripheral Chip Select Strobe Assert Computation Example . . . 771
Table 354. Peripheral Chip Select Strobe Negate Computation Example . . . 772
Table 355. DSPI Slave Mode System Constraints . . . 783
Table 356. Interrupt and DMA Request Conditions . . . 787
Table 357. Baud Rate Values (bps) . . . 791
Table 358. Delay Values . . . 792
Table 359. List of Terms . . . 795
Table 360. External Signal Properties . . . 800

Table 361. FlexRay memory map . . . 802
Table 362. Register Access Conventions . . . 806
Table 363. Additional Register Reset Conditions . . . 806
Table 364. Register Write Access Restrictions . . . 807
Table 365. FR_MVR Field Descriptions . . . 808
Table 366. FR_MCR Field Descriptions . . . 808
Table 367. FlexRay Channel Selection . . . 809
Table 368. FR_SYMBADR Field Descriptions . . . 810
Table 369. FR_STBSCR Field Descriptions . . . 811
Table 370. Strobe Signal Mapping . . . 811
Table 371. FR_MBDSR Field Descriptions . . . 812
Table 372. FR_MBSSUTR Field Descriptions . . . 813
Table 373. FR_PEDRAR Field Descriptions . . . 814
Table 374. FR_POCR Field Descriptions . . . 815
Table 375. FR_GIFER Field Descriptions . . . 817
Table 376. FR_PIFR0 Field Descriptions . . . 820
Table 377. FR_PIFR1 Field Descriptions . . . 822
Table 378. FR_PIER0 Field Descriptions . . . 823
Table 379. FR_PIER1 Field Descriptions . . . 824
Table 380. FR_CHIERFR Field Descriptions . . . 826
Table 381. FR_MBIVEC Field Descriptions . . . 828
Table 382. FR_CASERCR Field Descriptions . . . 828
Table 383. FR_CBSERCR Field Descriptions . . . 829
Table 384. FR_PSR0 Field Descriptions . . . 830
Table 385. FR_PSR1 Field Descriptions . . . 831
Table 386. FR_PSR2 Field Descriptions . . . 832
Table 387. FR_PSR3 Field Descriptions . . . 834
Table 388. FR_MTCTR Field Descriptions . . . 835
Table 389. FR_CYCTR Field Descriptions . . . 836
Table 390. FR_SLTCTAR Field Descriptions . . . 836
Table 391. FR_SLTCTBR Field Descriptions . . . 837
Table 392. FR_RTCORVR Field Descriptions . . . 837
Table 393. FR_OFCORVR Field Descriptions . . . 838
Table 394. FR_CIFR Field Descriptions . . . 839
Table 395. FR_SYMATOR Field Descriptions . . . 839
Table 396. FR_SFCNTR Field Descriptions . . . 840
Table 397. FR_SFTOR Field Description . . . 841
Table 398. FR_SFTCCSR Field Descriptions . . . 841
Table 399. FR_SFIDRFR Field Descriptions . . . 842
Table 400. FR_SFIDAFVR Field Descriptions . . . 843
Table 401. FR_SFIDAFMR Field Descriptions . . . 843
Table 402. NMVR[0:5] Field Descriptions . . . 844
Table 403. Mapping of NMVRn to the Received Payload Bytes NMVn . . . 844
Table 404. FR_NMVLR Field Descriptions . . . 845
Table 405. FR_TICCR Field Descriptions . . . 845
Table 406. FR_TI1CYSR Field Descriptions . . . 846
Table 407. FR_TI1MTOR Field Descriptions . . . 847
Table 408. FR_TI2CR0 Field Descriptions . . . 847
Table 409. FR_TI2CR1 Field Descriptions . . . 848
Table 410. FR_SSSR Field Descriptions . . . 849
Table 411. Mapping Between FR_SSSRn and FR_SSRn . . . 849
Table 412. FR_SSCCR Field Descriptions . . . 850

Table 413. Mapping between internal FR_SSCCRn and FR_SSCRn . . . 850
Table 414. FR_SSR0–FR_SSR7 Field Descriptions . . . 851
Table 415. FR_SSCR0–FR_SSCR3 Field Descriptions . . . 853
Table 416. FR_MTSACFR Field Descriptions . . . 853
Table 417. MTSBCFR Field Descriptions . . . 854
Table 418. FR_RSBIR Field Descriptions . . . 854
Table 419. FR_RFSDOR Field Descriptions . . . 855
Table 420. FR_RFSYMBADR Field Descriptions . . . 855
Table 421. FR_RFPTR Field Descriptions . . . 856
Table 422. SEL Controlled Receiver FIFO Registers . . . 856
Table 423. FR_RFWMSR Field Descriptions . . . 857
Table 424. FR_RFSIR Field Descriptions . . . 857
Table 425. RFDSR Field Descriptions . . . 858
Table 426. FR_RFARIR Field Descriptions . . . 858
Table 427. FR_RFBRIR Field Descriptions . . . 858
Table 428. FR_RFFLPCR Field Descriptions . . . 859
Table 429. FR_RFMIDAFVR Field Descriptions . . . 859
Table 430. FR_RFMIDAFMR Field Descriptions . . . 860
Table 431. FR_RFFIDRFVR Field Descriptions . . . 860
Table 432. FR_RFFIDRFMR Field Descriptions . . . 861
Table 433. FR_RFRFCFR Field Descriptions . . . 861
Table 434. FR_RFRFCTR Field Descriptions . . . 862
Table 435. FR_LDTXSLAR Field Descriptions . . . 863
Table 436. FR_LDTXSLBR Field Descriptions . . . 863
Table 437. Protocol Configuration Register Fields . . . 863
Table 438. Wakeup Channel Selection . . . 865
Table 439. FR_EEIFER Field Descriptions . . . 873
Table 440. FR_EERICR Field Descriptions . . . 875
Table 441. FR_EERAR Field Descriptions . . . 875
Table 442. LRAM Bank Value for MID = 1 . . . 876
Table 443. FR_EERDR Field Descriptions . . . 876
Table 444. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field . . . 876
Table 445. FR_EERSR Field Descriptions . . . 877
Table 446. FR_EEIAR Field Descriptions . . . 878
Table 447. FR_EEIDR Field Descriptions . . . 878
Table 448. FR_EEICR Field Descriptions . . . 879
Table 449. FR_MBCCSRn Field Descriptions . . . 880
Table 450. FR_MBCCFRn Field Descriptions . . . 881
Table 451. Channel assignment description . . . 881
Table 452. FR_MBFIDRn Field Descriptions . . . 882
Table 453. FR_MBIDXRn Field Descriptions . . . 882
Table 454. FR_MBDORn Field Descriptions . . . 883
Table 455. FR_LEETRn Field Descriptions . . . 883
Table 456. Frame Header Write Access Constraints (Transmit Message Buffer) . . . 896
Table 457. Frame Header Field Descriptions (Receive Message Buffer and Receive FFO) . . . 897
Table 458. Frame Header Field Descriptions (Transmit Message Buffer) . . . 897
Table 459. Receive Message Buffer Slot Status Content . . . 898
Table 460. Receive Message Buffer Slot Status Field Description . . . 899
Table 461. Transmit Message Buffer Slot Status Content . . . 900
Table 462. Transmit Message Buffer Slot Status Structure Field Descriptions . . . 901
Table 463. Message Buffer Data Field Minimum Length . . . 902
Table 464. Frame Data Write Access Constraints . . . 903

Table 465. Frame Data Field Descriptions . . . 903
Table 466. Individual Message Buffer Types . . . 904
Table 467. Transmit Message Buffer Access Regions Description . . . 906
Table 468. Transmit Message Buffer State Description . . . 907
Table 469. Transmit Message Buffer Application Transitions . . . 908
Table 470. Transmit Message Buffer Module Transitions . . . 908
Table 471. Transmit Message Buffer Transition Priorities . . . 909
Table 472. Receive Message Buffer Access Region Description . . . 914
Table 473. Receive Message Buffer States and Access . . . 914
Table 474. Receive Message Buffer Application Transitions . . . 916
Table 475. Receive Message Buffer Module Transitions . . . 916
Table 476. Receive Message Buffer Transition Priorities . . . 916
Table 477. Receive Message Buffer Update . . . 917
Table 478. Message Buffer Search Priority (static segment) . . . 920
Table 479. Message Buffer Search Priority (dynamic segment) . . . 920
Table 480. Sync Frame Table Generation Modes . . . 934
Table 481. Key Slot Frame Type . . . 936
Table 482. Slot Status Content . . . 940
Table 483. FlexRay Channel Bit Rate Control . . . 949
Table 484. PE DRAM Layout . . . 949
Table 485. CHI LRAM Layout . . . 951
Table 486. Detected Memory Error Types . . . 952
Table 487. PE DRAM checkbits coding . . . 953
Table 488. FR_EERCR[CODE] PE DRAM Syndrome Coding . . . 953
Table 489. CHI LRAM checkbits coding . . . 954
Table 490. FR_EERCR[CODE] CHI LRAM Syndrome Coding . . . 954
Table 491. Maximum SYMATOR[TIMEOUT] examples . . . 958
Table 492. Minimum f_{chi} [MHz] examples (128 message buffers used) . . . 961
Table 493. Protocol Control Command Priorities . . . 962
Table 494. Transmit Buffer Configuration . . . 963
Table 495. Receive Buffer Configuration . . . 963
Table 496. ECR[ETHER_EN] De-Assertion Effect on FEC . . . 968
Table 497. User Initialization (Before ENCTRL[ETHER_EN]) . . . 969
Table 498. FEC User Initialization (Before ECR[ETHER_EN]) . . . 969
Table 499. Microcontroller Initialization . . . 970
Table 500. MII Mode . . . 970
Table 501. 7-Wire Mode Configuration . . . 971
Table 502. Destination Address to 6-Bit Hash . . . 976
Table 503. PAUSE Frame Field Specification . . . 978
Table 504. Module Memory Map . . . 981
Table 505. FEC register map . . . 981
Table 506. MIB Counters Memory Map . . . 982
Table 507. EIR field descriptions . . . 985
Table 508. EIMR field descriptions . . . 987
Table 509. RDAR field descriptions . . . 987
Table 510. TDAR Field Descriptions . . . 988
Table 511. ECR field descriptions . . . 989
Table 512. MMFR field descriptions . . . 990
Table 513. MSCR field descriptions . . . 991
Table 514. Programming examples for MSCR . . . 991
Table 515. MIBC field descriptions . . . 991
Table 516. RCR field descriptions . . . 992

Table 517. TCR field descriptions . . . 993
Table 518. PALR field descriptions . . . 994
Table 519. PAUR field descriptions . . . 994
Table 520. OPD field descriptions . . . 995
Table 521. IAUR field descriptions . . . 995
Table 522. IALR field descriptions . . . 996
Table 523. GAUR field descriptions . . . 996
Table 524. GALR field descriptions . . . 997
Table 525. TFWR field descriptions . . . 997
Table 526. FRBR field descriptions . . . 998
Table 527. FRSR field descriptions . . . 998
Table 528. ERDSR field descriptions . . . 999
Table 529. ETDSR field descriptions . . . 999
Table 530. EMRBR field descriptions . . . 1000
Table 531. Receive Buffer Descriptor Field Definitions . . . 1002
Table 532. Transmit Buffer Descriptor Field Definitions . . . 1004
Table 533. eMIOS_0 channel to pin mapping . . . 1009
Table 534. eMIOS_1 channel to pin mapping . . . 1009
Table 535. STM memory map . . . 1011
Table 536. STM_CR field descriptions . . . 1012
Table 537. STM_CNT field descriptions . . . 1013
Table 538. STM_CCRn field descriptions . . . 1013
Table 539. STM_CIRn field descriptions . . . 1014
Table 540. STM_CMPn field descriptions . . . 1014
Table 541. eMIOS memory map . . . 1018
Table 542. Unified Channel memory map . . . 1019
Table 543. EMIOSMCR field descriptions . . . 1019
Table 544. Global prescaler clock divider . . . 1020
Table 545. EMIOSGFLAG field descriptions . . . 1021
Table 546. EMIOSOUDIS field descriptions . . . 1021
Table 547. EMIOSUCDIS field descriptions . . . 1022
Table 548. EMIOSA[n], EMIOSB[n] and EMIOSALTA[n] values assignment . . . 1023
Table 549. EMIOSC[n] field descriptions . . . 1025
Table 550. UC internalprescaler clock divider . . . 1026
Table 551. UC input filter bits . . . 1026
Table 552. UC BSL bits . . . 1027
Table 553. Channel mode selection . . . 1027
Table 554. EMIOSS[n] field descriptions . . . 1028
Table 555. PIT_RTI memory map . . . 1064
Table 556. Timer channel *n* / RTI channel . . . 1065
Table 557. PITMCR field descriptions . . . 1066
Table 558. LDVAL field descriptions . . . 1066
Table 559. CVAL field descriptions . . . 1067
Table 560. TCTRL field descriptions . . . 1068
Table 561. TFLG field descriptions . . . 1068
Table 562. ADC channel mapping . . . 1073
Table 563. ADC_0 mux control signal availability . . . 1075
Table 564. 10-bit ADC_0 digital registers . . . 1076
Table 565. 12-bit ADC_1 digital registers . . . 1081
Table 566. Main Configuration Register (MCR) field descriptions . . . 1084
Table 567. Main Status Register (MSR) field descriptions . . . 1086
Table 568. Interrupt Status Register (ISR) field descriptions . . . 1087

Table 569. CEOCFR[0..2] register description . . . 1087
Table 570. Channel Pending Registers (CEOCFR[0..2]) field descriptions . . . 1089
Table 571. Interrupt Mask Register (IMR) field descriptions . . . 1090
Table 572. CIMR[0..2] register description . . . 1090
Table 573. Channel Interrupt Mask Register (CIMR[0..2]) field descriptions . . . 1092
Table 574. Watchdog Threshold Interrupt Status Register (WTISR) field descriptions – ADC_1 . . . 1093
Table 575. Watchdog Threshold Interrupt Mask Register (WTIMR) field descriptions – ADC_0 . . . 1094
Table 576. Watchdog Threshold Interrupt Mask Register (WTIMR) field descriptions – ADC_1 . . . 1094
Table 577. DMA Enable Register (DMAE) field descriptions . . . 1095
Table 578. DMAR[0..2] register description . . . 1095
Table 579. DMA Channel Select Register (DMAR[0..2]) field descriptions . . . 1097
Table 580. Threshold Register field descriptions . . . 1098
Table 581. ADC_0 Threshold Register (THRHLR) field descriptions . . . 1098
Table 582. ADC_1 Threshold Register (THRHLR) field descriptions . . . 1098
Table 583. Presampling Control Register (PSCR) field descriptions . . . 1099
Table 584. PSR[0..2] register description . . . 1099
Table 585. Presampling Register (PSR[0..2]) field descriptions . . . 1101
Table 586. CTR[0..2] register description . . . 1101
Table 587. Conversion timing register field descriptions . . . 1102
Table 588. NCMR[0..2] register description . . . 1103
Table 589. Normal Conversion Mask Registers (NCMR[0..2]) field descriptions . . . 1104
Table 590. JCMR[0..2] register description . . . 1104
Table 591. Injected Conversion Mask Registers (JCMR[0..2]) field descriptions . . . 1106
Table 592. DSDR field descriptions . . . 1107
Table 593. Power-down Exit Delay Register (PDEDR) field descriptions . . . 1107
Table 594. CDR[0..95] register description . . . 1107
Table 595. Channel Data Register (CDR[0..95]) field descriptions . . . 1108
Table 596. CWSELR[0..11] register description . . . 1109
Table 597. CWSELR field descriptions – ADC_0 . . . 1110
Table 598. CWSELR1 field descriptions – ADC_0 . . . 1110
Table 599. CWSELR4 field descriptions – ADC_0 . . . 1111
Table 600. CWSELR5 field descriptions – ADC_0 . . . 1111
Table 601. CWSELR6 field descriptions – ADC_0 . . . 1111
Table 602. CWSELR7 field descriptions – ADC_0 . . . 1112
Table 603. CWSELR field descriptions – ADC_0 . . . 1112
Table 604. CWSELR9 field descriptions – ADC_0 . . . 1113
Table 605. CWSELR10 field descriptions – ADC_0 . . . 1113
Table 606. CWSELR11 field descriptions – ADC_0 . . . 1114
Table 607. CWSELR0 field descriptions – ADC_1 . . . 1114
Table 608. CWSELR1 field descriptions – ADC_1 . . . 1115
Table 609. CWSELR4 field descriptions – ADC_1 . . . 1115
Table 610. CWSELR5 field descriptions – ADC_1 . . . 1116
Table 611. CWENR[0..2] register description . . . 1116
Table 612. Channel Watchdog Enable Register (CWENRx, x = [0..2]) field descriptions . . . 1117
Table 613. Analog Watchdog Out of Range Register 0 (AWORR0) — ADC_0 and ADC_1 field description . . . 1118
Table 614. ADC sampling and conversion timing at 5 V / 3.3 V for ADC_0 . . . 1125
Table 615. Presampling voltage selection based on PREVALx fields . . . 1127
Table 616. Values of WDGxH and WDGxL fields . . . 1128
Table 617. CTU memory map . . . 1132
Table 618. CTU_EVTCFGRx field descriptions . . . 1132
Table 619. Trigger source . . . 1133

Table 620. CTU-to-ADC Channel Assignment . . . 1136
Table 621. Low power configuration . . . 1138
Table 622. SRAM memory map . . . 1138
Table 623. Number of wait states required for SRAM operations . . . 1140
Table 624. Flash memory multi module sectorization . . . 1145
Table 625. TestFlash structure . . . 1147
Table 626. Shadow sector structure . . . 1148
Table 627. Code flash memory user registers . . . 1150
Table 628. MCR field descriptions . . . 1152
Table 629. Array space size . . . 1155
Table 630. Low address space configuration . . . 1156
Table 631. Mid address space configuration . . . 1156
Table 632. MCR bits set/clear priority levels . . . 1156
Table 633. LML field descriptions . . . 1158
Table 634. HBL field descriptions . . . 1161
Table 635. SLL field descriptions . . . 1162
Table 636. LMS field descriptions . . . 1164
Table 637. HBS field descriptions . . . 1165
Table 638. ADR field descriptions . . . 1166
Table 639. ADR content: priority list . . . 1166
Table 640. BIU0 field descriptions . . . 1167
Table 641. BIU1 field descriptions . . . 1168
Table 642. BIU2 field descriptions . . . 1168
Table 643. BIU3 field descriptions . . . 1169
Table 644. UT0 field descriptions . . . 1170
Table 645. UT1 field descriptions . . . 1172
Table 646. UT2 field descriptions . . . 1173
Table 647. UMSIR0 field descriptions . . . 1173
Table 648. UMISR1 field descriptions . . . 1174
Table 649. UMISR2 field descriptions . . . 1175
Table 650. UMISR3 field descriptions . . . 1175
Table 651. UMISR4 field descriptions . . . 1176
Table 652. NVPWD0 field descriptions . . . 1177
Table 653. NVPWD1 field descriptions . . . 1178
Table 654. NVSCC0 field descriptions . . . 1179
Table 655. NVSCC1 field descriptions . . . 1180
Table 656. NVUSRO field descriptions . . . 1180
Table 657. NVUSRO_1 field descriptions . . . 1182
Table 658. Programming NVUSRO_1 and STCU fault grading parameters . . . 1182
Table 659. Interpretation of Flash read data . . . 1184
Table 660. Storing STCU data in shadow flash . . . 1184
Table 661. Flash memory modify operations . . . 1185
Table 662. Bits Manipulation: Double Words with the same ECC value . . . 1193
Table 663. 64 KB data flash memory module sectorization . . . 1198
Table 664. TestFlash structure . . . 1198
Table 665. Data Flash Single Bank Registers . . . 1200
Table 666. Abbreviations . . . 1201
Table 667. MCR field descriptions . . . 1202
Table 668. MCR bits set/clear priority levels . . . 1205
Table 669. LML field descriptions . . . 1206
Table 670. SLL field descriptions . . . 1208
Table 671. LMS field descriptions . . . 1210

Table 672. ADR field descriptions . . . 1211
Table 673. ADR content: priority list . . . 1211
Table 674. UT0 field descriptions . . . 1212
Table 675. UT1 field descriptions . . . 1214
Table 676. UMSIR0 field descriptions . . . 1215
Table 677. UMISR1 field descriptions . . . 1215
Table 678. Flash memory modify operations . . . 1216
Table 679. platform flash controller Module Connections . . . 1227
Table 680. Flash-related regions in the system memory map . . . 1227
Table 681. platform flash controller 32-bit Memory Map . . . 1230
Table 682. PFCR0 field descriptions . . . 1232
Table 683. PFCR1 field descriptions . . . 1236
Table 684. PFLASH Access Protection Register Field Descriptions . . . 1237
Table 685. Platform flash controller Stall-While-Write Interrupts . . . 1246
Table 686. Additional Wait-State Encoding . . . 1247
Table 687. Extended Additional Wait-State Encoding . . . 1247
Table 688. Register protection memory map . . . 1257
Table 689. SLBR*n* field descriptions . . . 1258
Table 690. Soft lock bits vs. protected address . . . 1258
Table 691. GCR field descriptions . . . 1259
Table 692. Protected registers . . . 1264
Table 693. SWT memory map . . . 1277
Table 694. SWT_CR Field Descriptions . . . 1278
Table 695. SWT_IR Field Descriptions . . . 1279
Table 696. SWT_TO Register Field Descriptions . . . 1279
Table 697. SWT_WN Register Field Descriptions . . . 1280
Table 698. SWT_SR Field Descriptions . . . 1280
Table 699. SWT_CO Register Field Descriptions . . . 1281
Table 700. SWT_SR Field Descriptions . . . 1281
Table 701. ECSM memory map . . . 1285
Table 702. PCT field descriptions . . . 1286
Table 703. REV field descriptions . . . 1286
Table 704. IMC field descriptions . . . 1287
Table 705. MIR field descriptions . . . 1288
Table 706. MUDCR field descriptions . . . 1289
Table 707. ECR field descriptions . . . 1290
Table 708. ESR field descriptions . . . 1292
Table 709. EEGR field descriptions . . . 1293
Table 710. PFEAR field descriptions . . . 1295
Table 711. PFEMR field descriptions . . . 1296
Table 712. PFEAT field descriptions . . . 1296
Table 713. PFEDR field descriptions . . . 1297
Table 714. PREAR field descriptions . . . 1298
Table 715. PRESR field descriptions . . . 1298
Table 716. RAM syndrome mapping for single-bit correctable errors . . . 1298
Table 717. PREMR field descriptions . . . 1300
Table 718. PREAT field descriptions . . . 1301
Table 719. PREDR field descriptions . . . 1302
Table 720. Acronyms and abbreviated terms . . . 1303
Table 721. STCU registers configuration after the boot sequence phase 1 . . . 1308
Table 722. STCU register map . . . 1312
Table 723. STCU_SKC field descriptions . . . 1314

Table 724. STCU_CFG field descriptions . . . 1315
Table 725. STCU_WDGG field descriptions . . . 1316
Table 726. STCU_CRCE field descriptions . . . 1317
Table 727. STCU_CRCR field descriptions . . . 1317
Table 728. STCU_ERR field descriptions . . . 1318
Table 729. STCU_ERRK field descriptions . . . 1320
Table 730. STCU_MBSL field descriptions . . . 1320
Table 731. STCU_MBSH field descriptions . . . 1321
Table 732. STCU MBIST status bits to partition mapping . . . 1322
Table 733. STCU_MBEL field descriptions . . . 1324
Table 734. STCU_MBEH field descriptions . . . 1325
Table 735. STCU_MBSEK field descriptions . . . 1325
Table 736. STCU_MBCFML field descriptions . . . 1326
Table 737. STCU_MBCFMH field descriptions . . . 1327
Table 738. STCU_MBSFML field descriptions . . . 1327
Table 739. STCU_MBSFMH field descriptions . . . 1328
Table 740. STCU_MBFMK field descriptions . . . 1329
Table 741. STCU_MB_CTRL field descriptions . . . 1330
Table 742. CSE memory map . . . 1333
Table 743. CSE_CR field descriptions . . . 1335
Table 744. CSE_SR field descriptions . . . 1336
Table 745. CSE_IR field descriptions . . . 1337
Table 746. CSE_ECR field descriptions . . . 1338
Table 747. CSE_CMD field descriptions . . . 1339
Table 748. CSE_Px field descriptions . . . 1339
Table 749. Memory Slots . . . 1341
Table 750. Memory Slot Security Flags . . . 1342
Table 751. Error Code Summary . . . 1344
Table 752. ENC_ECB Command . . . 1346
Table 753. ENC_CBC Command . . . 1346
Table 754. DEC_ECB Command . . . 1346
Table 755. DEC_CBC Command . . . 1347
Table 756. GENERATE_MAC Command . . . 1347
Table 757. VERIFY_MAC Command . . . 1348
Table 758. LOAD_KEY Command . . . 1348
Table 759. LOAD_PLAIN_KEY Command . . . 1349
Table 760. EXPORT_RAM_KEY Command . . . 1349
Table 761. INIT_RNG Command . . . 1350
Table 762. EXTEND_SEED Command . . . 1350
Table 763. RND Command . . . 1351
Table 764. SECURE_BOOT Command . . . 1351
Table 765. BOOT_FAILURE Command . . . 1351
Table 766. BOOT_OK Command . . . 1352
Table 767. GET_ID Command . . . 1352
Table 768. CANCEL Command . . . 1352
Table 769. DEBUG_CHAL Command . . . 1353
Table 770. DEBUG_AUTH Command . . . 1353
Table 771. TRNG_RND Command . . . 1353
Table 772. INIT_CSE Command . . . 1354
Table 773. JTAG Signal Properties . . . 1357
Table 774. JTAG Instructions . . . 1362
Table 775. Signal Properties . . . 1370

Table 776. Nexus Client JTAG Instructions . . . 1371
Table 777. NDI Registers . . . 1371
Table 778. NDI Configuration Options . . . 1374
Table 779. MCKO_DIV Values . . . 1374
Table 780. NPC Memory Map . . . 1377
Table 781. DID field descriptions . . . 1378
Table 782. PCR field descriptions . . . 1379
Table 783. NPC Output Messages . . . 1382
Table 784. Loading NEXUS-ENABLE instruction . . . 1385
Table 785. Write to a 32-Bit Nexus Client Register . . . 1386
Table 786. Terms and Definitions . . . 1389
Table 787. Public TCODEs supported . . . 1390
Table 788. Error Code Encoding (TCODE = 8) . . . 1393
Table 789. Error Type Encoding (TCODE = 8) . . . 1394
Table 790. RCODE values (TCODE = 27) . . . 1394
Table 791. Event Code Encoding (TCODE = 33) . . . 1394
Table 792. Data Trace Size Encodings (TCODE = 5,6,13,14) . . . 1395
Table 793. Nexus3 Registers . . . 1395
Table 794. DC1 Field Descriptions . . . 1397
Table 795. DC2 Field Descriptions . . . 1398
Table 796. DC3 Field Descriptions . . . 1401
Table 797. DC4 Field Descriptions . . . 1402
Table 798. DS Field Descriptions . . . 1402
Table 799. RWCS Field Description . . . 1403
Table 800. Read/Write Access Status Bit Encoding . . . 1404
Table 801. RWD Data Placement for Transfers . . . 1405
Table 802. RWD data placement for Transfers . . . 1405
Table 803. WT Field Descriptions . . . 1407
Table 804. Program Trace Start Trigger Control Register Fields . . . 1408
Table 805. Program Trace End Trigger Control Register Fields . . . 1409
Table 806. Data Trace Start Trigger Control Register Fields . . . 1410
Table 807. Data Trace End Trigger Control Register Fields . . . 1411
Table 808. Watchpoint Mask Register Fields . . . 1412
Table 809. Overrun Control Register Fields . . . 1413
Table 810. DTC Field Description . . . 1414
Table 811. Data Trace—Address Range Options . . . 1415
Table 812. Nexus3 Debug Requirements . . . 1417
Table 813. Document revision history . . . 1419

List of figures

Figure 1. Register figure conventions . . . 66
Figure 2. SPC564Bxx and SPC56ECxx block diagram . . . 75
Figure 3. 176-pin LQFP configuration . . . 101
Figure 4. 208-pin LQFP configuration . . . 102
Figure 5. LBGA208 configuration . . . 103
Figure 6. 256-pin BGA configuration . . . 104
Figure 7. Boot mode selection . . . 141
Figure 8. Boot sector structure . . . 142
Figure 9. Flash memory boot mode sequence . . . 144
Figure 10. Censorship control in flash memory boot mode . . . 148
Figure 11. Censorship control in serial boot mode . . . 149
Figure 12. BAM logic flow . . . 150
Figure 13. BAM censorship mode detection . . . 153
Figure 14. BAM serial boot mode flow for censorship enabled and private password . . . 155
Figure 15. Start address, VLE bit and download size in bytes . . . 156
Figure 16. LINFlexD bit timing in UART mode . . . 157
Figure 17. FlexCAN bit timing . . . 159
Figure 18. SSCM block diagram . . . 160
Figure 19. System Status register (SSCM_STATUS) . . . 162
Figure 20. System Memory Configuration register (SSCM_MEMCONFIG) . . . 163
Figure 21. Error Configuration (SSCM_ERROR) . . . 164
Figure 22. Debug Status Port Register (SSCM_DEBUGPORT) . . . 165
Figure 23. Password Comparison Register High Word (SSCM_PWCMPH) . . . 166
Figure 24. Password Comparison Register Low Word (SSCM_PWCMPL) . . . 167
Figure 25. DPM Boot register (SSCM_DPMBOOT) . . . 168
Figure 26. DPM Boot Key register (SSCM_DPMKEY) . . . 168
Figure 27. User Option Status register (SSCM_UOPS) . . . 169
Figure 28. Processor Start Address register (SSCM_PSA) . . . 170
Figure 29. Code Length register (SSCM_CLEN) . . . 170
Figure 30. SPC564Bxx and SPC56ECxx system clock generation . . . 173
Figure 31. Fast External Crystal Oscillator Control Register (FXOSC_CTL) . . . 177
Figure 32. Slow External Crystal Oscillator Control register (SXOSC_CTL) . . . 179
Figure 33. Low Power RC Control Register (SIRC_CTL) . . . 181
Figure 34. FIRC Oscillator Control Register (FIRC_CTL) . . . 183
Figure 35. FMPLL block diagram . . . 184
Figure 36. Control Register (CR) . . . 186
Figure 37. Modulation Register (MR) . . . 188
Figure 38. FMPLL output clock division flow during progressive switching . . . 190
Figure 39. Frequency modulation . . . 191
Figure 40. Clock Monitor Unit diagram . . . 193
Figure 41. Control Status Register (CMU_CSR) . . . 196
Figure 42. Frequency Display Register (CMU_FDR) . . . 197
Figure 43. High Frequency Reference Register FMPLL (CMU_HFREFR) . . . 197
Figure 44. Low Frequency Reference Register FMPLL (CMU_LFREFR) . . . 198
Figure 45. Interrupt status register (CMU_ISR) . . . 198
Figure 46. Measurement Duration Register (CMU_MDR) . . . 199
Figure 47. MC_CGM block diagram . . . 201
Figure 48. e200z0h Core Clock Divider Configuration Register (CGM_Z0_DCR) . . . 208

Figure 49. FEC Clock Divider Configuration Register (CGM_FEC_DCR) . . . 208
Figure 50. Flash Clock Divider Configuration Register (CGM_Flash_DCR) . . . 209
Figure 51. Output Clock Enable Register (CGM_OC_EN) . . . 209
Figure 52. Output Clock Division Select Register (CGM_OCDS_SC) . . . 210
Figure 53. System Clock Select Status Register (CGM_SC_SS) . . . 211
Figure 54. System Clock Divider 0 Configuration Register (CGM_SC_DC0) . . . 211
Figure 55. System Clock Divider 1 Configuration Register (CGM_SC_DC1) . . . 212
Figure 56. System Clock Divider 2 Configuration Register (CGM_SC_DC2) . . . 212
Figure 57. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) . . . 213
Figure 58. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) . . . 214
Figure 59. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0) . . . 214
Figure 60. MC_CGM System Clock Generation Overview . . . 216
Figure 61. MC_CGM Auxiliary Clock 0 Generation Overview . . . 217
Figure 62. MC_CGM Auxiliary Clock 1 Generation Overview . . . 218
Figure 63. MC_CGM Output Clock Multiplexer and PA[0] Generation . . . 219
Figure 64. MC_ME block diagram . . . 221
Figure 65. Global Status Register (ME_GS) . . . 233
Figure 66. Mode Control Register (ME_MCTL) . . . 236
Figure 67. Mode Enable Register (ME_ME) . . . 237
Figure 68. Interrupt Status Register (ME_IS) . . . 238
Figure 69. Interrupt Mask Register (ME_IM) . . . 240
Figure 70. Invalid Mode Transition Status Register (ME_IMTS) . . . 241
Figure 71. Debug Mode Transition Status Register (ME_DMTS) . . . 242
Figure 72. RESET Mode Configuration Register (ME_RESET_MC) . . . 245
Figure 73. TEST Mode Configuration Register (ME_TEST_MC) . . . 245
Figure 74. SAFE Mode Configuration Register (ME_SAFE_MC) . . . 246
Figure 75. DRUN Mode Configuration Register (ME_DRUN_MC) . . . 246
Figure 76. RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC) . . . 247
Figure 77. HALT Mode Configuration Register (ME_HALT_MC) . . . 248
Figure 78. STOP Mode Configuration Register (ME_STOP_MC) . . . 248
Figure 79. STANDBY Mode Configuration Register (ME_STANDBY_MC) . . . 249
Figure 80. Peripheral Status Register 0 (ME_PS0) . . . 252
Figure 81. Peripheral Status Register 1 (ME_PS1) . . . 253
Figure 82. Peripheral Status Register 2 (ME_PS2) . . . 253
Figure 83. Peripheral Status Register 3 (ME_PS3) . . . 254
Figure 84. Run Peripheral Configuration Registers (ME_RUN_PC0...7) . . . 254
Figure 85. Low-Power Peripheral Configuration Registers (ME_LP_PC0...7) . . . 255
Figure 86. Peripheral Control Registers (ME_PCTLn) . . . 256
Figure 87. MC_ME mode diagram . . . 260
Figure 88. MC_ME transition diagram . . . 272
Figure 89. Interaction between ME_PCTLn, ME_LPn, and ME_RUN_PCn . . . 276
Figure 90. MC_ME Application example flow diagram . . . 277
Figure 91. MC_RGM block diagram . . . 279
Figure 92. Functional Event Status Register (RGM_FES) . . . 284
Figure 93. Destructive Event Status Register (RGM_DES) . . . 286
Figure 94. Functional Event Reset Disable Register (RGM_FERD) . . . 287
Figure 95. Destructive Event Reset Disable Register (RGM_DERD) . . . 288
Figure 96. Functional Event Alternate Request Register (RGM_FEAR) . . . 289
Figure 97. Functional Event Short Sequence Register (RGM_FESS) . . . 290
Figure 98. STANDBY Reset Sequence Register (RGM_STDBY) . . . 292
Figure 99. Functional Bidirectional Reset Enable Register (RGM_FBRE) . . . 293
Figure 100. MC_RGM State Machine . . . 296

Figure 101. MC_PCU block diagram . . . 302
Figure 102. Power Domain #0 Configuration Register (PCU_PCONF0) . . . 305
Figure 103. Power Domain #1 Configuration Register (PCU_PCONF1) . . . 306
Figure 104. Power Domain #2…3 Configuration Registers (PCU_PCONF2…3) . . . 307
Figure 105. Power Domain Status Register (PCU_PSTAT) . . . 308
Figure 106. MC_PCU Events During Power Sequences (non-STANDBY mode) . . . 309
Figure 107. MC_PCU Events During Power Sequences (STANDBY mode) . . . 310
Figure 108. Voltage Regulator Control Register (VREG_CTL) . . . 315
Figure 109. Voltage Regulator Register (VREG_PDMODE) . . . 315
Figure 110. Power domain organization . . . 317
Figure 111. Wakeup unit block diagram . . . 320
Figure 112. Wake-up port mapping . . . 322
Figure 113. NMI Status Flag Register (NSR) . . . 323
Figure 114. NMI Configuration Register (NCR) . . . 324
Figure 115. Wakeup/Interrupt Status Flag Register (WISR) . . . 325
Figure 116. Interrupt Request Enable Register (IRER) . . . 326
Figure 117. Wakeup Request Enable Register (WRER) . . . 326
Figure 118. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER) . . . 326
Figure 119. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER) . . . 327
Figure 120. Wakeup/Interrupt Filter Enable Register (WIFER) . . . 327
Figure 121. Wakeup/Interrupt Pullup Enable Register (WIPUER) . . . 328
Figure 122. NMI pad diagram . . . 329
Figure 123. External Interrupt Pad Diagram . . . 330
Figure 124. RTC/API block diagram . . . 333
Figure 125. Clock gating for RTC clocks . . . 334
Figure 126. RTC Supervisor Control Register (RTCSUPV) . . . 335
Figure 127. RTC Control Register (RTCC) . . . 336
Figure 128. RTC Status Register (RTCS) . . . 338
Figure 129. RTC Counter Register (RTCCNT) . . . 339
Figure 130. Extended CAN data frame . . . 341
Figure 131. Control Register (CR) . . . 342
Figure 132. CAN Sampler Sample Registers 0–11 . . . 343
Figure 133. e200z0h block diagram . . . 348
Figure 134. e200z0h Supervisor mode programmer’s model . . . 351
Figure 135. e200z0h User mode program model . . . 352
Figure 136. e200z4d Block Diagram . . . 355
Figure 137. e200z4d Supervisor Mode Programmer’s Model . . . 360
Figure 138. e200z4d User mode programmer’s model SPRs . . . 361
Figure 139. DMA block diagram . . . 366
Figure 140. DMA Control Register (EDMA_CR) . . . 371
Figure 141. DMA Error Status (EDMA_ESR) Register . . . 374
Figure 142. DMA Enable Request (EDMA_ERQRL) Registers . . . 376
Figure 143. DMA Enable Error Interrupt (EDMA_EEIRL) Register . . . 377
Figure 144. DMA Set Enable Request (EDMA_SERQR) Register . . . 377
Figure 145. DMA Clear Enable Request (EDMA_CERQR) Register . . . 378
Figure 146. DMA Set Enable Error Interrupt (EDMA_SEEIR) Register . . . 378
Figure 147. DMA Clear Enable Error Interrupt (EDMA_CEEIR) Register . . . 379
Figure 148. DMA Clear Interrupt Request (EDMA_CIRQR) Fields . . . 379
Figure 149. DMA Clear Error (EDMA_CER) Register . . . 380
Figure 150. DMA Set START Bit (EDMA_SSBR) Register . . . 380
Figure 151. DMA Clear DONE Status (EDMA_CDSBR) Register . . . 381
Figure 152. DMA Interrupt Request (EDMA_IRQRL) Registers . . . 381

Figure 153. DMA Error (EDMA_ERL) Registers . . . 382
Figure 154. DMA Hardware Request Status (EDMA_HRSL) Register . . . 383
Figure 155. DMA Channel n Priority (EDMA_CPRn) Register. . . . 384
Figure 156. TCD Structure . . . 385
Figure 157. eDMA Operation, Part 1 . . . 393
Figure 158. eDMA Operation, Part 2 . . . 394
Figure 159. eDMA Operation, Part 3 . . . 395
Figure 160. Example of Multiple Loop Iterations . . . 400
Figure 161. Memory array terms . . . 400
Figure 162. DMA_MUX block diagram . . . 412
Figure 163. Channel Configuration Registers (CHCONFIG#n) . . . 414
Figure 164. DMA Mux triggered channels . . . 418
Figure 165. DMA Mux Channel Triggering: Normal Operation . . . 418
Figure 166. DMA Mux Channel Triggering: Ignored Trigger . . . 419
Figure 167. INTC block diagram . . . 428
Figure 168. INTC Software Vector mode . . . 429
Figure 169. Program Flow–Software Vector mode . . . 429
Figure 170. e200z0h Software Vector Mode: Interrupt Exception Handler Address Calculation . . . 430
Figure 171. e200z4d Software Vector Mode: Interrupt Exception Handler Address Calculation . . . 430
Figure 172. Program Flow–Hardware Vector mode . . . 431
Figure 173. e200z0h Hardware Vector Mode: Interrupt Exception Handler Address Calculation . . . 432
Figure 174. e200z4d Hardware Vector Mode: Interrupt Exception Handler Address Calculation . . . 432
Figure 175. INTC Module Configuration Register (INTC_MCR) . . . 434
Figure 176. INTC Current Priority Register for Processor 0 (e200z4d) (INTC_CPR_PRC0) . . . 435
Figure 177. INTC Current Priority Register for Processor 1 (e200z0h) (INTC_CPR_PRC1) . . . 436
Figure 178. INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0) when INTC_MCR[VTES] = 0 . . . 436
Figure 179. INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0) when INTC_MCR[VTES] = 1 . . . 437
Figure 180. INTC Interrupt Acknowledge Register for Processor 1 (e200z0h) (INTC_IACKR_PRC1) when INTC_MCR[VTES] = 0 . . . 438
Figure 181. INTC Interrupt Acknowledge Register for Processor 1 (e200z0h) (INTC_IACKR_PRC1) when INTC_MCR[VTES] = 1 . . . 438
Figure 182. INTC End of Interrupt Register for Processor 0 (e200z4d) (INTC_EOIR_PRC0). . . . 439
Figure 183. INTC End of Interrupt Register for processor 1 (e200z0h) (INTC_EOIR_PRC1). . . . 439
Figure 184. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3]). . . . 439
Figure 185. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7]). . . . 440
Figure 186. INTC Priority Select Register 0–3 (INTC_PSR[0:3]). . . . 440
Figure 187. INTC Priority Select Register 276-278 (INTC_PSR[276:278]) . . . 441
Figure 188. Cores NMI sources . . . 460
Figure 189. Software Vector Mode Handshaking Timing Diagram . . . 464
Figure 190. Hardware Vector Mode Handshaking Timing Diagram. . . . 465
Figure 191. Raised Priority Preserved Timing Diagram. . . . 471
Figure 192. XBAR block diagram. . . . 476
Figure 193. Master Priority Registers (MPRn) . . . 479
Figure 194. Slave General Purpose Control Register n (SGPCRn). . . . 481
Figure 195. Master General Purpose Control Register n (MGPCRn) . . . 482
Figure 196. Low to high priority mastership change . . . 490
Figure 197. High to low priority mastership change. . . . 491
Figure 198. Round-robin mastership change. . . . 491
Figure 199. Parking on a specific master. . . . 493
Figure 200. Parking on last master . . . 493

Figure 201. MPU block diagram . . . 498
Figure 202. MPU Control/Error Status Register (MPU_CESR) . . . 502
Figure 203. MPU Error Address Register, Slave Port n (MPU_EARn) . . . 504
Figure 204. MPU Error Detail Register, Slave Port n (MPU_EDRn) . . . 504
Figure 205. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0) . . . 506
Figure 206. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1) . . . 506
Figure 207. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2) . . . 508
Figure 208. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3) . . . 511
Figure 209. MPU RGD Alternate Access Control n (MPU_RGDAACn) . . . 512
Figure 210. MPU access evaluation macro . . . 515
Figure 211. Semaphores block diagram . . . 519
Figure 212. SEMA4 Gate n Register (SEMA4_GATEn) . . . 522
Figure 213. Semaphores Processor *n* IRQ Notification Enable (SEMA4_CP{0,1}INE) . . . 523
Figure 214. Semaphores Processor *n* IRQ Notification (SEMA4_CP{0,1}NTF) . . . 523
Figure 215. Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT) . . . 524
Figure 216. Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF) . . . 526
Figure 217. Branch Unit Control and Status Register (BUCSR) . . . 532
Figure 218. L1 Cache Control and Status Register 1 (L1CSR1) . . . 535
Figure 219. MMU Assist Register 2 (MAS2) . . . 537
Figure 220. Influence of compiler settings on application performance and code size . . . 538
Figure 221. Machine State Register (MSR) . . . 539
Figure 222. System Integration Unit Lite block diagram . . . 543
Figure 223. MCU ID Register #1 (MIDR1) . . . 547
Figure 224. MCU ID Register #2 (MIDR2) . . . 548
Figure 225. Interrupt Status Flag Register (ISR) . . . 549
Figure 226. Interrupt Request Enable Register (IRER) . . . 549
Figure 227. Interrupt Rising-Edge Event Enable Register (IREER) . . . 550
Figure 228. Interrupt Falling-Edge Event Enable Register (IFEER) . . . 550
Figure 229. Interrupt Filter Enable Register (IFER) . . . 551
Figure 230. Pad Configuration Registers (PCRx) . . . 552
Figure 231. Pad Selection for Multiplexed Inputs Register (PSMI0_3) . . . 554
Figure 232. Port GPIO Pad Data Output Register 0–3 (GPDO0_3) . . . 560
Figure 233. Port GPIO Pad Data Input Register 0–3 (GPDI0_3) . . . 560
Figure 234. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23) . . . 564
Figure 235. Interrupt Filter Clock Prescaler Register (IFCPR) . . . 565
Figure 236. Parallel Input Select Register 0 (PISR0) . . . 566
Figure 237. Parallel Input Select Register 1 (PISR1) . . . 566
Figure 238. Parallel Input Select Register 2 (PISR2) . . . 566
Figure 239. Parallel Input Select Register 3 (PISR3) . . . 567
Figure 240. Parallel Input Select Register 4 (PISR4) . . . 567
Figure 241. Parallel Input Select Register 5 (PISR5) . . . 567
Figure 242. Parallel Input Select Register 6 (PISR6) . . . 568
Figure 243. Parallel Input Select Register 7 (PISR7) . . . 568
Figure 244. Parallel Input Select Register 8 (PISR8) . . . 568
Figure 245. Parallel Input Select Register 9 (PISR9) . . . 569
Figure 246. Parallel Input Select Register 10 (PISR10) . . . 569
Figure 247. Parallel Input Select Register 11 (PISR11) . . . 569
Figure 248. Parallel Input Select Register 12 (PISR12) . . . 570
Figure 249. Parallel Input Select Register 13 (PISR13) . . . 570
Figure 250. Parallel Input Select Register 14 (PISR14) . . . 570
Figure 251. Parallel Input Select Register 15 (PISR15) . . . 571
Figure 252. eMIOS_0 to DSPI_0 serialization . . . 571

Figure 253. DSPI Input Select Register (DISR) . . . 575
Figure 254. Data Port example arrangement showing configuration for different port width accesses 578
Figure 255. External interrupt pad diagram . . . 579
Figure 256. I2C block diagram . . . 581
Figure 257. I2C Bus Address Register (IBAD) . . . 582
Figure 258. I2C Bus Frequency Divider Register (IBFD) . . . 582
Figure 259. SDA hold time . . . 584
Figure 260. SCL divider and SDA hold . . . 584
Figure 261. I2C Bus Control Register (IBCR) . . . 589
Figure 262. I2C Bus Status Register (IBSR) . . . 590
Figure 263. I2C Bus Data I/O Register (IBDR) . . . 591
Figure 264. I2C Bus Interrupt Config Register (IBIC) . . . 592
Figure 265. I2C module DMA interface block diagram . . . 593
Figure 266. I2C bus transmission signals . . . 594
Figure 267. Start and stop conditions . . . 595
Figure 268. I2C bus clock synchronization . . . 597
Figure 269. Flow-Chart of Typical I2C Interrupt Routine . . . 602
Figure 270. DMA mode master transmit . . . 604
Figure 271. DMA mode master receive . . . 605
Figure 272. LINFlexD block diagram . . . 608
Figure 273. LIN network topology . . . 610
Figure 274. LIN frame structure . . . 611
Figure 275. Break field . . . 611
Figure 276. Sync pattern . . . 612
Figure 277. Structure of the data field . . . 612
Figure 278. Identifier . . . 612
Figure 279. LINFlexD controller operating modes . . . 614
Figure 280. Filter configuration - register organization . . . 620
Figure 281. Identifier match index . . . 621
Figure 282. LIN sync field measurement . . . 622
Figure 283. LINFlexD in Loop Back mode . . . 623
Figure 284. LINFlexD in Self Test mode . . . 624
Figure 285. UART mode 8-bit data frame . . . 624
Figure 286. UART mode 9-bit data frame . . . 625
Figure 287. UART mode 16-bit data frame . . . 625
Figure 288. UART mode 17-bit data frame . . . 625
Figure 289. LIN control register 1 (LINCR1) . . . 631
Figure 290. LIN interrupt enable register (LINIER) . . . 634
Figure 291. LIN status register (LINSR) . . . 635
Figure 292. LIN error status register (LINESR) . . . 638
Figure 293. UART mode control register (UARTCR) . . . 639
Figure 294. UART mode status register (UARTSR) . . . 642
Figure 295. LIN timeout control status register (LINTCSR) . . . 644
Figure 296. LIN output compare register (LINOCR) . . . 645
Figure 297. LIN timeout control register (LINTOCR) . . . 645
Figure 298. LIN timeout control register (LINTOCR) . . . 646
Figure 299. LIN integer baud rate register (LINIBRR) . . . 646
Figure 300. LIN checksum field register (LINCFR) . . . 647
Figure 301. LIN control register 2 (LINCR2) . . . 648
Figure 302. Buffer identifier register (BIDR) . . . 649
Figure 303. Buffer data register least significant (BDRL) . . . 650
Figure 304. Buffer data register most significant (BDRM) . . . 651

Figure 305. Identifier filter enable register (IFER) . . . 651
Figure 306. Identifier filter match index (IFMI) . . . 652
Figure 307. Identifier filter mode register (IFMR) . . . 653
Figure 308. Identifier filter control registers (IFCR0–IFCR15) . . . 654
Figure 309. Global control register (GCR) . . . 655
Figure 310. UART preset timeout register (UARTPTO) . . . 656
Figure 311. UART current timeout register (UARTCTO) . . . 657
Figure 312. DMA Tx enable register (DMATXE) . . . 657
Figure 313. DMA Rx enable register (DMARXE) . . . 658
Figure 314. TCD chain memory map (master node, TX mode) . . . 659
Figure 315. FSM to control the DMA TX interface (master node) . . . 661
Figure 316. TCD chain memory map (master node, RX mode) . . . 662
Figure 317. FSM to control the DMA RX interface (master node) . . . 663
Figure 318. TCD chain memory map (slave node, TX mode) . . . 664
Figure 319. FSM to control the DMA TX interface (slave node) . . . 666
Figure 320. TCD chain memory map (slave node, RX mode) . . . 667
Figure 321. FSM to control the DMA RX interface (slave node) . . . 669
Figure 322. TCD chain memory map (UART node, TX mode) . . . 670
Figure 323. FSM to control the DMA TX interface (UART node) . . . 671
Figure 324. TCD chain memory map (UART node, RX mode) . . . 672
Figure 325. FSM to control the DMA RX interface (UART node) . . . 674
Figure 326. Header and response timeout . . . 677
Figure 327. Interrupt diagram . . . 679
Figure 328. Programming consideration: master node, transmitter . . . 681
Figure 329. Programming consideration: master node, receiver . . . 681
Figure 330. Programming consideration: master node, transmitter, bit error . . . 681
Figure 331. Programming consideration: master node, receiver, checksum error . . . 681
Figure 332. Programming consideration: slave node, transmitter, no filters . . . 682
Figure 333. Programming consideration: slave node, receiver, no filters . . . 682
Figure 334. Programming consideration: slave node, transmitter, no filters, bit error . . . 682
Figure 335. Programming consideration: slave node, receiver, no filters, checksum error . . . 682
Figure 336. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter . . . 683
Figure 337. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter . . . 683
Figure 338. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset . . . 683
Figure 339. Programming consideration: slave node, TX filter, BF is set . . . 684
Figure 340. Programming consideration: slave node, RX filter, BF is set . . . 684
Figure 341. Programming consideration: slave node, TX filter, RX filter, BF is set . . . 685
Figure 342. Programming consideration: extended frames . . . 685
Figure 343. Programming consideration: response timeout(1) . . . 686
Figure 344. Programming consideration: frame timeout . . . 686
Figure 345. Programming consideration: header timeout . . . 686
Figure 346. Programming consideration: UART mode . . . 686
Figure 347. FlexCAN block diagram . . . 688
Figure 348. Message Buffer Structure . . . 693
Figure 349. Rx FIFO structure . . . 697
Figure 350. ID Table 0–7 . . . 697
Figure 351. Module Configuration Register (MCR) . . . 699
Figure 352. Control Register (CTRL) . . . 703
Figure 353. Free Running Timer (TIMER) . . . 706

Figure 354. Rx Global Mask Register (RXGMASK) . . . 707
Figure 355. Error Counter Register (ECR) . . . 709
Figure 356. Error and Status Register (ESR) . . . 710
Figure 357. Interrupt Masks 2 Register (IMASK2) . . . 713
Figure 358. Interrupt Masks 1 Register (IMASK1) . . . 713
Figure 359. Interrupt Flags 2 Register (IFLAG2) . . . 714
Figure 360. Interrupt Flags 1 Register (IFLAG1) . . . 715
Figure 361. Rx Individual Mask Registers (RXIMR0–RXIMR63) . . . 716
Figure 362. CAN Engine Clocking Scheme . . . 723
Figure 363. Segments within the Bit Time . . . 724
Figure 364. Arbitration, Match and Move Time Windows . . . 725
Figure 365. DSPI block diagram . . . 730
Figure 366. DSPI with Queues and DMA . . . 732
Figure 367. DSPI Module Configuration Register (DSPI_MCR) . . . 737
Figure 368. DSPI Transfer Count Register (DSPI_TCR) . . . 739
Figure 369. DSPI Clock and Transfer Attributes Register 0–5 (DSPI_CTAR0–DSPI_CTAR5) in the master mode . . . 741
Figure 370. DSPI Clock and Transfer Attributes Register 0, 1 (DSPI_CTAR0, DSPI_CTAR1) in the slave mode . . . 741
Figure 371. DSPI Status Register (DSPI_SR) . . . 746
Figure 372. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) . . . 748
Figure 373. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode . . . 749
Figure 374. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode . . . 750
Figure 375. DSPI POP RX FIFO Register (DSPI_POPR) . . . 751
Figure 376. DSPI Transmit FIFO Register 0–3 (DSPI_TXFR0–DSPI_TXFR3) . . . 752
Figure 377. DSPI Receive FIFO Registers 0–3 (DSPI_RXFR0–DSPI_RXFR3) . . . 752
Figure 378. DSPI DSI Configuration Register (DSPI_DSICR) . . . 753
Figure 379. DSPI DSI Serialization Data Register (DSPI_SDR) . . . 755
Figure 380. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR) . . . 755
Figure 381. DSPI DSI Transmit Comparison Register (DSPI_COMPR) . . . 756
Figure 382. DSPI Deserialization Data Register (DSPI_DDR) . . . 756
Figure 383. DSPI DSI Configuration Register 1 (DSPI_DSICR1) . . . 757
Figure 384. DSPI DSI Serialization Source Select Register (DSPI_SSR) . . . 758
Figure 385. DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR) . . . 758
Figure 386. DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR) . . . 759
Figure 387. SPI and DSI Serial Protocol Overview . . . 760
Figure 388. DSI Serialization Diagram . . . 765
Figure 389. DSI Deserialization Diagram . . . 766
Figure 390. Example of Parallel Chaining of DSPIs . . . 767
Figure 391. Example of Serial Chaining of DSPIs . . . 768
Figure 392. Communications Clock Prescalers and Scalers . . . 769
Figure 393. Peripheral Chip Select Strobe Timing . . . 771
Figure 394. DSPI Transfer Timing Diagram (MTFE=0, CPHA=0, FMSZ=8) . . . 773
Figure 395. DSPI Transfer Timing Diagram (MTFE=0, CPHA=1, FMSZ=8) . . . 774
Figure 396. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/4$) . . . 776
Figure 397. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/2$) . . . 776
Figure 398. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/3$) . . . 777
Figure 399. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/2$) . . . 778
Figure 400. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/3$) . . . 778
Figure 401. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/4$) . . . 779
Figure 402. Example of Non-Continuous Format (CPHA=1, CONT=0) . . . 780
Figure 403. Example of Continuous Transfer (CPHA=1, CONT=1) . . . 780

Figure 404. Continuous SCK Timing Diagram (CONT=0) . . . 782
Figure 405. Continuous SCK Timing Diagram (CONT=1) . . . 782
Figure 406. DSPI usage in the TSB Configuration . . . 784
Figure 407. TSB Downstream Frames . . . 784
Figure 408. TSB Data Frame Format for MSC Dual Receiver Operation . . . 786
Figure 409. TX FIFO Pointers and Counter . . . 793
Figure 410. FLEXRAY Block Diagram . . . 797
Figure 411. Module Version Register (FR_MVR) . . . 807
Figure 412. Module Configuration Register (FR_MCR) . . . 808
Figure 413. System Memory Base Address High Register (FR_SYMBADHR) . . . 810
Figure 414. System Memory Base Address Low Register (FR_SYMBADLR) . . . 810
Figure 415. Strobe Signal Control Register (FR_STBSCR) . . . 811
Figure 416. Message Buffer Data Size Register (FR_MBDSR) . . . 812
Figure 417. Message Buffer Segment Size and Utilization Register (FR_MBSSUTR) . . . 813
Figure 418. PE DRAM Access Register (FR_PEDRAR) . . . 813
Figure 419. PE DRAM Data Register (FR_PEDRDR) . . . 814
Figure 420. Protocol Operation Control Register (FR_POCR) . . . 814
Figure 421. Global Interrupt Flag and Enable Register (FR_GIFER) . . . 817
Figure 422. Protocol Interrupt Flag Register 0 (FR_PIFR0) . . . 819
Figure 423. Protocol Interrupt Flag Register 1 (FR_PIFR1) . . . 821
Figure 424. Protocol Interrupt Enable Register 0 (FR_PIER0) . . . 822
Figure 425. Protocol Interrupt Enable Register 1 (FR_PIER1) . . . 824
Figure 426. CHI Error Flag Register (FR_CHIERFR) . . . 825
Figure 427. Message Buffer Interrupt Vector Register (FR_MBIVEC) . . . 827
Figure 428. Channel A Status Error Counter Register (FR_CASERCR) . . . 828
Figure 429. Channel B Status Error Counter Register (FR_CBSERCR) . . . 828
Figure 430. Protocol Status Register 0 (FR_PSR0) . . . 829
Figure 431. Protocol Status Register 1 (FR_PSR1) . . . 831
Figure 432. Protocol Status Register 2 (FR_PSR2) . . . 832
Figure 433. Protocol Status Register 3 (FR_PSR3) . . . 834
Figure 434. Macrotick Counter Register (FR_MTCTR) . . . 835
Figure 435. Cycle Counter Register (FR_CYCTR) . . . 836
Figure 436. Slot Counter Channel A Register (FR_SLTCTAR) . . . 836
Figure 437. Slot Counter Channel B Register (FR_SLTCTBR) . . . 836
Figure 438. Rate Correction Value Register (FR_RTCORVR) . . . 837
Figure 439. Offset Correction Value Register (FR_OFCORVR) . . . 837
Figure 440. Combined Interrupt Flag Register (FR_CIFR) . . . 838
Figure 441. System Memory Access Time-Out Register (FR_SYMATOR) . . . 839
Figure 442. Sync Frame Counter Register (FR_SFCNTR) . . . 840
Figure 443. Sync Frame Table Offset Register (FR_SFTOR) . . . 840
Figure 444. Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR) . . . 841
Figure 445. Sync Frame ID Rejection Filter Register (FR_SFIDRFR) . . . 842
Figure 446. Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR) . . . 843
Figure 447. Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR) . . . 843
Figure 448. Network Management Vector Registers (FR_NMVR0–FR_NMVR5) . . . 844
Figure 449. Network Management Vector Length Register (FR_NMVLR) . . . 845
Figure 450. Timer Configuration and Control Register (FR_TICCR) . . . 845
Figure 451. Timer 1 Cycle Set Register (FR_TI1CYSR) . . . 846
Figure 452. Timer 1 Macrotick Offset Register (FR_TI1MTOR) . . . 847
Figure 453. Timer 2 Configuration Register 0 (FR_TI2CR0) . . . 847
Figure 454. Timer 2 Configuration Register 1 (FR_TI2CR1) . . . 848
Figure 455. Slot Status Selection Register (FR_SSSR) . . . 848

Figure 456. Slot Status Counter Condition Register (FR_SSCCR) . . . 849
Figure 457. Slot Status Registers (FR_SSR0–FR_SSR7). . . . 851
Figure 458. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3) . . . 852
Figure 459. MTS A Configuration Register (FR_MTSACFR). . . . 853
Figure 460. MTS B Configuration Register (MTSBCFR) . . . 853
Figure 461. Receive Shadow Buffer Index Register (FR_RSBIR). . . . 854
Figure 462. Receive FIFO Start Data Offset Register (FR_RFSDOR) . . . 855
Figure 463. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR) . . . 855
Figure 464. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR). . . . 855
Figure 465. Receive FIFO Periodic Timer Register (FR_RFPTR). . . . 856
Figure 466. Receive FIFO Watermark and Selection Register (FR_RFWMSR) . . . 856
Figure 467. Receive FIFO Start Index Register (FR_RFSIR) . . . 857
Figure 468. Receive FIFO Depth and Size Register (RFDSR) . . . 857
Figure 469. Receive FIFO A Read Index Register (FR_RFARIR). . . . 858
Figure 470. Receive FIFO B Read Index Register (FR_RFBRIR). . . . 858
Figure 471. Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) . . . 859
Figure 472. Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR) . . . 859
Figure 473. Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR) . . . 860
Figure 474. Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR) . . . 860
Figure 475. Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR) . . . 860
Figure 476. Receive FIFO Range Filter Configuration Register (FR_RFRFCFR) . . . 861
Figure 477. Receive FIFO Range Filter Control Register (FR_RFRFCTR). . . . 861
Figure 478. Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR) . . . 862
Figure 479. Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR) . . . 863
Figure 480. Protocol Configuration Register 0 (FR_PCR0) . . . 865
Figure 481. Protocol Configuration Register 1 (FR_PCR1) . . . 866
Figure 482. Protocol Configuration Register 2 (FR_PCR2) . . . 866
Figure 483. Protocol Configuration Register 3 (FR_PCR3) . . . 866
Figure 484. Protocol Configuration Register 4 (FR_PCR4) . . . 866
Figure 485. Protocol Configuration Register 5 (FR_PCR5) . . . 866
Figure 486. Protocol Configuration Register 6 (FR_PCR6) . . . 867
Figure 487. Protocol Configuration Register 7 (FR_PCR7) . . . 867
Figure 488. Protocol Configuration Register 8 (FR_PCR8) . . . 867
Figure 489. Protocol Configuration Register 9 (FR_PCR9) . . . 867
Figure 490. Protocol Configuration Register 10 (FR_PCR10) . . . 868
Figure 491. Protocol Configuration Register 11 (FR_PCR11) . . . 868
Figure 492. Protocol Configuration Register 12 (FR_PCR12) . . . 868
Figure 493. Protocol Configuration Register 13 (FR_PCR13) . . . 868
Figure 494. Protocol Configuration Register 14 (FR_PCR14) . . . 869
Figure 495. Protocol Configuration Register 15 (FR_PCR15) . . . 869
Figure 496. Protocol Configuration Register 16 (FR_PCR16) . . . 869
Figure 497. Protocol Configuration Register 17 (FR_PCR17) . . . 869
Figure 498. Protocol Configuration Register 18 (FR_PCR18) . . . 869
Figure 499. Protocol Configuration Register 19 (FR_PCR19) . . . 870
Figure 500. Protocol Configuration Register 20 (FR_PCR20) . . . 870
Figure 501. Protocol Configuration Register 21 (FR_PCR21) . . . 870
Figure 502. Protocol Configuration Register 22 (FR_PCR22) . . . 870
Figure 503. Protocol Configuration Register 23 (FR_PCR23) . . . 870
Figure 504. Protocol Configuration Register 24 (FR_PCR24) . . . 871
Figure 505. Protocol Configuration Register 25 (FR_PCR25) . . . 871
Figure 506. Protocol Configuration Register 26 (FR_PCR26) . . . 871
Figure 507. Protocol Configuration Register 27 (FR_PCR27) . . . 871

Figure 508. Protocol Configuration Register 28 (FR_PCR28) . . . 872
Figure 509. Protocol Configuration Register 29 (FR_PCR29) . . . 872
Figure 510. Protocol Configuration Register 30 (FR_PCR30) . . . 872
Figure 511. ECC Error Interrupt Flag and Enable Register (FR_EEIFER) . . . 872
Figure 512. ECC Error Report and Injection Control Register (FR_EERICR) . . . 874
Figure 513. ECC Error Report Address Register (FR_EERAR) . . . 875
Figure 514. ECC Error Report Data Register (FR_EERDR) . . . 876
Figure 515. ECC Error Report Code Register (FR_EERCR) . . . 877
Figure 516. ECC Error Injection Address Register (FR_EEIAR) . . . 878
Figure 517. ECC Error Injection Data Register (FR_EEIDR) . . . 878
Figure 518. ECC Error Injection Code Register (FR_EEICR) . . . 879
Figure 519. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) . . . 879
Figure 520. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn) . . . 881
Figure 521. Message Buffer Frame ID Registers (FR_MBFIDRn) . . . 882
Figure 522. Message Buffer Index Registers (FR_MBIDXRn) . . . 882
Figure 523. Message Buffer Data Field Offset Registers (FR_MBDORn) . . . 883
Figure 524. LRAM ECC Error Test Registers (FR_LEETRn) . . . 883
Figure 525. Physical Message Buffer Structure . . . 884
Figure 526. Individual Message Buffer Structure . . . 886
Figure 527. Receive Shadow Buffer Structure . . . 887
Figure 528. Receive FIFO Structure . . . 889
Figure 529. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 0) . . . 892
Figure 530. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 1) . . . 893
Figure 531. Frame Header Structure (Receive Message Buffer and Receive FIFO) . . . 895
Figure 532. Frame Header Structure (Transmit Message Buffer) . . . 895
Figure 533. Frame Header Structure (Transmit Message Buffer for Key Slot) . . . 896
Figure 534. Receive Message Buffer Slot Status Structure (ChAB) . . . 898
Figure 535. Receive Message Buffer Slot Status Structure (ChA) . . . 898
Figure 536. Receive Message Buffer Slot Status Structure (ChB) . . . 898
Figure 537. Transmit Message Buffer Slot Status Structure (ChAB) . . . 900
Figure 538. Transmit Message Buffer Slot Status Structure (ChA) . . . 900
Figure 539. Transmit Message Buffer Slot Status Structure (ChB) . . . 901
Figure 540. Message Buffer Data Field Structure . . . 902
Figure 541. Transmit Message Buffer Access Regions . . . 905
Figure 542. Transmit Message Buffer States . . . 906
Figure 543. Message Transmission Timing . . . 910
Figure 544. Message Transmission from HLck state with unlock . . . 910
Figure 545. Null Frame Transmission from Idle state . . . 911
Figure 546. Null Frame Transmission from HLck state . . . 911
Figure 547. Null Frame Transmission from HLck state with unlock . . . 911
Figure 548. Null Frame Transmission from Idle state with locking . . . 912
Figure 549. Receive Message Buffer Access Regions . . . 913
Figure 550. Receive Message Buffer States . . . 914
Figure 551. Message Reception Timing . . . 918
Figure 552. Inconsistent Channel Assignment . . . 921
Figure 553. Message Buffer Reconfiguration Scheme . . . 923
Figure 554. Received Frame FIFO Filter Path . . . 927
Figure 555. Dual Channel Device Mode . . . 930
Figure 556. Single Channel Device Mode (Channel A) . . . 931
Figure 557. Single Channel Device Mode (Channel B) . . . 931
Figure 558. External Offset Correction Write and Application Timing . . . 932
Figure 559. External Rate Correction Write and Application Timing . . . 932

Figure 560. Sync Table Memory Layout . . . 933
Figure 561. Sync Frame Table Trigger and Generation Timing. . . . 935
Figure 562. Strobe Signal Timing (type = pulse, clk_offset = -2) . . . 938
Figure 563. Strobe Signal Timing (type = pulse, clk_offset = +4) . . . 938
Figure 564. Slot Status Vector Update . . . 940
Figure 565. Slot Status Counting and FR_SSCRn Update . . . 942
Figure 566. Scheme of FR_GIFER interrupt signal generation . . . 946
Figure 567. Scheme of FR_EEIFER interrupt signal generation . . . 947
Figure 568. Scheme of FR_CIFR flags generation . . . 948
Figure 569. Transmit Data Not Available . . . 964
Figure 570. Transmit Data Available . . . 964
Figure 571. FEC Block Diagram . . . 967
Figure 572. Ethernet Address Recognition—Receive Block Decisions . . . 974
Figure 573. Ethernet Address Recognition—Microcode Decisions . . . 975
Figure 574. Ethernet Interrupt Event Register (EIR) . . . 985
Figure 575. Ethernet Interrupt Mask Register (EIMR) . . . 986
Figure 576. Receive Descriptor Active Register (RDAR). . . . 987
Figure 577. Transmit Descriptor Active Register (TDAR) . . . 988
Figure 578. Ethernet Control Register (ECR) . . . 988
Figure 579. MII Management Frame Register (MMFR). . . . 990
Figure 580. MII Speed Control Register (MSCR) . . . 990
Figure 581. MIB Control Register (MIBC) . . . 991
Figure 582. Receive Control Register (RCR). . . . 992
Figure 583. Transmit Control Register (TCR) . . . 993
Figure 584. Physical Address Lower Register (PALR) . . . 994
Figure 585. Physical Address Upper Register (PAUR) . . . 994
Figure 586. Opcode/Pause Duration Register (OPD) . . . 995
Figure 587. Descriptor Individual Upper Address Register (IAUR) . . . 995
Figure 588. Descriptor Individual Lower Address Register (IALR). . . . 996
Figure 589. Descriptor Group Upper Address Register (GAUR) . . . 996
Figure 590. Descriptor Group Lower Address Register (GALR) . . . 996
Figure 591. Transmit FIFO Watermark Register (TFWR). . . . 997
Figure 592. FIFO Receive Bound Register (FRBR). . . . 997
Figure 593. FIFO Receive Bound Register (FRBR). . . . 998
Figure 594. FIFO Receive Start Register (FRSR) . . . 998
Figure 595. Receive Descriptor Ring Start Register (ERDSR) . . . 999
Figure 596. Transmit Buffer Descriptor Ring Start Register (ETDSR). . . . 999
Figure 597. Receive Buffer Size Register (EMRBR) . . . 1000
Figure 598. Receive Buffer Descriptor (RxBD) . . . 1002
Figure 599. Transmit Buffer Descriptor (TxBD) . . . 1004
Figure 600. Interaction between timers and relevant peripherals . . . 1007
Figure 601. STM Control Register (STM_CR) . . . 1012
Figure 602. STM Count Register (STM_CNT). . . . 1013
Figure 603. STM Channel Control Register (STM_CCRn) . . . 1013
Figure 604. STM Channel Interrupt Register (STM_CIRn) . . . 1014
Figure 605. STM Channel Compare Register (STM_CMPn). . . . 1014
Figure 606. Channel configuration. . . . 1017
Figure 607. eMIOS Module Configuration Register (EMIOSMCR) . . . 1019
Figure 608. eMIOS Global FLAG (EMIOSGFLAG) Register . . . 1021
Figure 609. eMIOS Output Update Disable (EMIOSOUDIS) Register. . . . 1021
Figure 610. eMIOS Enable Channel (EMIOSUCDIS) Register . . . 1022
Figure 611. eMIOS UC A Register (EMIOSA[n]) . . . 1022

Figure 612. eMIOS UC B Register (EMIOSB[n]) . . . 1023
Figure 613. eMIOS UC Counter Register (EMIOSCNT[n]) . . . 1024
Figure 614. eMIOS UC Control Register (EMIOSC[n]) . . . 1024
Figure 615. eMIOS UC Status Register (EMIOSS[n]) . . . 1028
Figure 616. eMIOS UC Alternate A register (EMIOSALTA[n]) . . . 1029
Figure 617. Single action input capture with rising edge triggering example . . . 1031
Figure 618. Single action input capture with both edges triggering example . . . 1031
Figure 619. SAOC example with EDPOL value being transferred to the output flip-flop . . . 1032
Figure 620. SAOC example toggling the output flip-flop . . . 1032
Figure 621. SAOC example with flag behavior . . . 1033
Figure 622. Input pulse width measurement example . . . 1034
Figure 623. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads . . . 1034
Figure 624. Input period measurement example . . . 1035
Figure 625. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads . . . 1036
Figure 626. Double action output compare with FLAG set on the second match . . . 1037
Figure 627. Double action output compare with FLAG set on both matches . . . 1037
Figure 628. DAOC with transfer disabling example . . . 1038
Figure 629. Modulus Counter Up mode example . . . 1039
Figure 630. Modulus Counter Up/Down mode example . . . 1040
Figure 631. Modulus Counter Buffered (MCB) Up Count mode . . . 1041
Figure 632. Modulus Counter Buffered (MCB) Up/Down mode . . . 1041
Figure 633. MCB Mode A1 Register Update in Up Counter mode . . . 1042
Figure 634. MCB Mode A1 Register Update in Up/Down Counter mode . . . 1042
Figure 635. OPWFMB A1 and B1 match to Output Register Delay . . . 1043
Figure 636. OPWFMB Mode with A1 = 0 (0% duty cycle) . . . 1044
Figure 637. OPWFMB A1 and B1 registers update and flags . . . 1045
Figure 638. OPWFMB mode from 100% to 0% duty cycle . . . 1045
Figure 639. OPWMCB A1 and B1 registers load . . . 1047
Figure 640. OPWMCB with lead dead time insertion . . . 1048
Figure 641. OPWMCB with trail dead time insertion . . . 1049
Figure 642. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3) . . . 1050
Figure 643. OPWMB mode matches and flags . . . 1052
Figure 644. OPWMB mode with 0% duty cycle . . . 1053
Figure 645. OPWMB mode from 100% to 0% duty cycle . . . 1053
Figure 646. OPWMT example . . . 1056
Figure 647. OPWMT with 0% Duty Cycle . . . 1056
Figure 648. OPWMT with 100% duty cycle . . . 1057
Figure 649. Input programmable filter submodule diagram . . . 1057
Figure 650. Input programmable filter example . . . 1058
Figure 651. Time base period when running in the fastest prescaler ratio . . . 1060
Figure 652. Time base generation with external clock and clear on match start . . . 1061
Figure 653. Time base generation with internal clock and clear on match start . . . 1061
Figure 654. Time base generation with clear on match end . . . 1062
Figure 655. PIT_RTI block diagram . . . 1063
Figure 656. PIT_RTI Module Control Register (PITMCR) . . . 1065
Figure 657. Timer Load Value Register (LDVAL) . . . 1066
Figure 658. Current Timer Value Register (CVAL) . . . 1067
Figure 659. Timer Control Register (TCTRL) . . . 1067
Figure 660. Timer Flag Register (TFLG) . . . 1068
Figure 661. Stopping and starting a timer . . . 1069
Figure 662. Modifying running timer period . . . 1069
Figure 663. Dynamically setting a new load value . . . 1069

Figure 664. Implementation of ADC_0 and ADC_1 . . . 1072
Figure 665. Main Configuration Register (MCR) – ADC_0 . . . 1083
Figure 666. Main Configuration Register (MCR) – ADC_1 . . . 1084
Figure 667. Main Status Register (MSR) . . . 1086
Figure 668. Interrupt Status Register (ISR) . . . 1087
Figure 669. Channel Pending Register 0 (CEOCFR0) — ADC_0 and ADC_1 . . . 1088
Figure 670. Channel Pending Register 1 (CEOCFR1) – ADC_0 . . . 1088
Figure 671. Channel Pending Register 1 (CEOCFR1) – ADC_1 . . . 1089
Figure 672. Channel Pending Register 2 (CEOCFR2) — ADC_0 . . . 1089
Figure 673. Interrupt Mask Register (IMR) . . . 1090
Figure 674. Channel Interrupt Mask Register 0 (CIMR0) — ADC_0 and ADC_1 . . . 1091
Figure 675. Channel Interrupt Mask Register 1 (CIMR1) – ADC_0 . . . 1091
Figure 676. Channel Interrupt Mask Register 1 (CIMR1) – ADC_1 . . . 1091
Figure 677. Channel Interrupt Mask Register 2 (CIMR2) — ADC_0 . . . 1092
Figure 678. Watchdog Threshold Interrupt Status Register (WTISR) – ADC_0 . . . 1092
Figure 679. Watchdog Threshold Interrupt Status Register (WTISR) – ADC_1 . . . 1093
Figure 680. Watchdog Threshold Interrupt Mask Register (WTIMR) – ADC_0 . . . 1093
Figure 681. Watchdog Threshold Interrupt Mask Register (WTIMR) – ADC_1 . . . 1094
Figure 682. DMA Enable Register (DMAE) . . . 1095
Figure 683. DMA Channel Select Register 0 (DMAR0) — ADC_0 and ADC_1 . . . 1096
Figure 684. DMA Channel Select Register 1 (DMAR1) – ADC_0 . . . 1096
Figure 685. DMA Channel Select Register 1 (DMAR1) – ADC_1 . . . 1096
Figure 686. DMA Channel Select Register 2 (DMAR2) — ADC_0 . . . 1097
Figure 687. Threshold Register – ADC_0 . . . 1097
Figure 688. ADC_1 Threshold Register THRHLR – ADC_1 . . . 1098
Figure 689. Presampling Control Register (PSCR) . . . 1099
Figure 690. Presampling Register 0 (PSR0) — ADC_0 and ADC_1 . . . 1100
Figure 691. Presampling Register 1 (PSR1) – ADC_0 . . . 1100
Figure 692. Presampling Register 1 (PSR1) – ADC_1 . . . 1100
Figure 693. Presampling Register 2 (PSR2) — ADC_0 . . . 1101
Figure 694. Conversion timing register . . . 1102
Figure 695. Normal Conversion Mask Register 0 (NCMR0) — ADC_0 and ADC_1 . . . 1103
Figure 696. Normal Conversion Mask Register 1 (NCMR1) – ADC_0 . . . 1103
Figure 697. Normal Conversion Mask Register 1 (NCMR1) – ADC_1 . . . 1104
Figure 698. Normal Conversion Mask Register 2 (NCMR2) — ADC_0 . . . 1104
Figure 699. Injected Conversion Mask Register 0 (JCMR0) — ADC_0 and ADC_1 . . . 1105
Figure 700. Injected Conversion Mask Register 1 (JCMR1) – ADC_0 . . . 1105
Figure 701. Injected Conversion Mask Register 1 (JCMR1) — ADC_1 . . . 1105
Figure 702. Injected Conversion Mask Register 2 (JCMR2) — ADC_0 . . . 1106
Figure 703. Decode Signals Delay Register (DSDR) . . . 1106
Figure 704. Power-down Exit Delay Register (PDEDR) . . . 1107
Figure 705. Channel Data Register (CDR[0..95]) — ADC_0 . . . 1108
Figure 706. Channel Data Register (CDR[0..95]) — ADC_1 . . . 1109
Figure 707. Channel Watchdog Select Register 0 (CWSELR0) – ADC_0 . . . 1109
Figure 708. Channel Watchdog Select Register 1 (CWSELR1) – ADC_0 . . . 1110
Figure 709. Channel Watchdog Select Register 4 (CWSELR4) – ADC_0 . . . 1110
Figure 710. Channel Watchdog Select Register 5 (CWSELR5) – ADC_0 . . . 1111
Figure 711. Channel Watchdog Select Register 6 (CWSELR6) – ADC_0 . . . 1111
Figure 712. Channel Watchdog Select Register 7 (CWSELR7) – ADC_0 . . . 1112
Figure 713. Channel Watchdog Select Register 8 (CWSELR8) – ADC_0 . . . 1112
Figure 714. Channel Watchdog Select Register 9 (CWSELR9) – ADC_0 . . . 1113
Figure 715. Channel Watchdog Select Register 10 (CWSELR10) – ADC_0 . . . 1113

Figure 716. Channel Watchdog Select Register 11 (CWSELR11) – ADC_0. 1114
Figure 717. Channel Watchdog Select Register 0 (CWSELR0) – ADC_1. 1114
Figure 718. Channel Watchdog Select Register 1 (CWSELR1) – ADC_1. 1115
Figure 719. Channel Watchdog Select Register 4 (CWSELR4) – ADC_1. 1115
Figure 720. Channel Watchdog Select Register 5 (CWSELR5) – ADC_1. 1116
Figure 721. Channel Watchdog Enable Register 0 (CWENR0) — ADC_0 and ADC_1 1116
Figure 722. Channel Watchdog Enable Register 1 (CWENR1) – ADC_0 1117
Figure 723. Channel Watchdog Enable Register 1 (CWENR1) – ADC_1 1117
Figure 724. Channel Watchdog Enable Register 2 (CWENR2) — ADC_0 1117
Figure 725. Analog Watchdog Out of Range Register 0 (AWORR0) — ADC_0 and ADC_1 1118
Figure 726. Analog Watchdog Out of Range Register 1 (AWORR1) – ADC_0. 1119
Figure 727. Analog Watchdog Out of Range Register 1 (AWORR1) – ADC_1. 1119
Figure 728. Analog Watchdog Out of Range Register 2 (AWORR2) — ADC_0 1120
Figure 729. Normal conversion flow 1121
Figure 730. Injected sample/conversion sequence 1122
Figure 731. Prescaler simplified block diagram 1124
Figure 732. Presampling sequence 1127
Figure 733. Presampling sequence with PRECONV = 1. 1127
Figure 734. Guarded area 1128
Figure 735. Cross Triggering Unit block diagram 1131
Figure 736. Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63) 1132
Figure 737. Flash memory architecture 1142
Figure 738. Flash memory module structure 1144
Figure 739. Module Configuration Register (MCR) 1151
Figure 740. Low/Mid address space block Locking register (LML) 1157
Figure 741. High address space Block Locking register (HBL) 1160
Figure 742. Secondary Low/mid address space block Locking register (SLL) 1161
Figure 743. Low/Mid address space block Select register (LMS) 1164
Figure 744. High address space Block Select register (HBS) 1165
Figure 745. Address Register (ADR) 1166
Figure 746. Bus Interface Unit 0 register (BIU0) 1167
Figure 747. Bus Interface Unit 1 register (BIU1) 1167
Figure 748. Bus Interface Unit 2 register (BIU2) 1168
Figure 749. Bus Interface Unit 0 register (BIU0) 1169
Figure 750. User Test 0 register (UT0) 1170
Figure 751. User Test 1 register (UT1) 1172
Figure 752. User Test 2 register (UT2) 1172
Figure 753. User Multiple Input Signature Register 0 (UMISR0). 1173
Figure 754. User Multiple Input Signature Register 1 (UMISR1). 1174
Figure 755. User Multiple Input Signature Register 2 (UMISR2). 1174
Figure 756. User Multiple Input Signature Register 3 (UMISR3). 1175
Figure 757. User Multiple Input Signature Register 4(UMISR4) 1176
Figure 758. Nonvolatile private censorship PassWord 0 register (NVPWD0) 1177
Figure 759. Nonvolatile private censorship PassWord 1 register (NVPWD1) 1178
Figure 760. Nonvolatile System Censoring Information 0 register (NVSCC0). 1178
Figure 761. Nonvolatile System Censoring Information 1 register (NVSCC1). 1179
Figure 762. Nonvolatile User Options register (NVUSRO) 1180
Figure 763. Nonvolatile User Options register 1(NVUSRO_1) 1181
Figure 764. Data flash memory module structure 1197
Figure 765. Module Configuration Register (MCR) 1201
Figure 766. Nonvolatile Low/Mid address space block Locking register (NVLML) 1206
Figure 767. Nonvolatile Secondary Low/mid address space block Locking reg (NVSLL) 1208

Figure 768. Low/Mid address space block Select register (LMS) . . . 1210
Figure 769. Address Register (ADR) . . . 1211
Figure 770. User Test 0 register (UT0) . . . 1212
Figure 771. User Test 1 register (UT1) . . . 1214
Figure 772. User Multiple Input Signature Register 0 (UMISR0) . . . 1214
Figure 773. User Multiple Input Signature Register 1 (UMISR1) . . . 1215
Figure 774. Code Flash1 Array Address Decode . . . 1229
Figure 775. PFLASH Configuration Register 0 (PFCR0) . . . 1231
Figure 776. PFLASH Configuration Register 1 (PFCR1) . . . 1235
Figure 777. PFLASH Access Protection Register (PFAPR) . . . 1237
Figure 778. 1-Cycle Access, No Buffering, No Prefetch . . . 1248
Figure 779. 3-Cycle Access, No Prefetch, Buffering Disabled . . . 1249
Figure 780. 3-Cycle Access, No Prefetch, Buffering Enabled . . . 1250
Figure 781. 3-Cycle Access, Prefetch and Buffering Enabled . . . 1251
Figure 782. 3-Cycle Access, Stall-and-Retry with Bn_RWWC = 11x . . . 1252
Figure 783. 3-Cycle Access, Abort-and-Retry with Bn_RWWC = 10x . . . 1253
Figure 784. Register Protection Module Block Diagram . . . 1254
Figure 785. Memory map of module using Register Protection . . . 1256
Figure 786. Soft Lock Bit Register (SLBR*n*) . . . 1258
Figure 787. Global Configuration Register (GCR) . . . 1259
Figure 788. Change Lock Settings Directly Via Area #4 . . . 1260
Figure 789. Change Lock Settings for 16-bit Protected Addresses . . . 1261
Figure 790. Change Lock Settings for 32-bit Protected Addresses . . . 1261
Figure 791. Change Lock Settings for Mixed Protection . . . 1262
Figure 792. Enable Locking Via Mirror Module Space (Area #3) . . . 1262
Figure 793. Enable Locking for Protected and Unprotected Addresses . . . 1263
Figure 794. SWT Control Register (SWT_CR) . . . 1277
Figure 795. SWT Interrupt Register (SWT_IR) . . . 1279
Figure 796. SWT Time-Out Register (SWT_TO) . . . 1279
Figure 797. SWT Window Register (SWT_WN) . . . 1280
Figure 798. SWT Service Register (SWT_SR) . . . 1280
Figure 799. SWT Counter Output Register (SWT_CO) . . . 1281
Figure 800. SWT Service Register (SWT_SR) . . . 1281
Figure 801. Pseudorandom Key Generator . . . 1282
Figure 802. Processor Core Type Register (PCT) . . . 1286
Figure 803. SoC-Defined Platform Revision Register (REV) . . . 1286
Figure 804. IPS On-Platform Module Configuration Register (IMC) . . . 1287
Figure 805. Miscellaneous Interrupt (MIR) Register . . . 1287
Figure 806. Miscellaneous User-Defined Control (MUDCR) Register . . . 1288
Figure 807. ECC Configuration (ECR) Register . . . 1290
Figure 808. ECC Status Register (ESR) . . . 1291
Figure 809. ECC Error Generation Register (EEGR) . . . 1293
Figure 810. Platform Flash ECC Address Register (PFEAR) . . . 1295
Figure 811. Platform Flash ECC Master Number Register (PFEMR) . . . 1295
Figure 812. Platform Flash ECC Attributes Register (PFEAT) . . . 1296
Figure 813. Platform Flash ECC Data Register (PFEDR) . . . 1297
Figure 814. Platform RAM ECC Address Register (PREAR) . . . 1297
Figure 815. Platform RAM ECC Syndrome Register (PRESR) . . . 1298
Figure 816. Platform RAM ECC Master Number Register (PREMR) . . . 1300
Figure 817. Platform RAM ECC Attributes Register (PREAT) . . . 1301
Figure 818. Platform RAM ECC Data Register (PREDR) . . . 1301
Figure 819. STCU block diagram . . . 1304

Figure 820. The STCU within the Safety Integrity Subsystem . . . 1306
Figure 821. Boot sequence phase 1: Self test . . . 1307
Figure 822. Boot sequence phase 2: Functional reset . . . 1307
Figure 823. STCU SK Code Register (STCU_SKC) . . . 1314
Figure 824. STCU Configuration Register (STCU_CFG) . . . 1315
Figure 825. STCU Watchdog Register Granularity (STCU_WDGG) . . . 1316
Figure 826. STCU CRC Expected Status Register (STCU_CRCE) . . . 1316
Figure 827. STCU CRC Read Status Register (STCU_CRCR) . . . 1317
Figure 828. STCU Error Register (STCU_ERR) . . . 1318
Figure 829. STCU Error Key Register (STCU_ERRK) . . . 1320
Figure 830. STCU MBIST Status Low Register (STCU_MBSL) . . . 1320
Figure 831. STCU MBIST Status High Register (STCU_MBSH) . . . 1321
Figure 832. STCU MBIST End Flag Low Register (STCU_MBEL) . . . 1324
Figure 833. STCU MBIST End Flag High Register (STCU_MBEH) . . . 1324
Figure 834. STCU MBIST Status-End Key Register (STCU_MBSEK) . . . 1325
Figure 835. STCU MBIST Critical FM Low Register (STCU_MBCFML) . . . 1326
Figure 836. STCU MBIST Critical FM High Register (STCU_MBCFMH) . . . 1326
Figure 837. STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML) . . . 1327
Figure 838. STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH) . . . 1328
Figure 839. STCU MBIST FM Key Register (STCU_MBFMK) . . . 1329
Figure 840. STCU MBIST Control Register (STCU_MB_CTRL) . . . 1330
Figure 841. CSE block diagram . . . 1333
Figure 842. CSE Control Register (CSE_CR) . . . 1334
Figure 843. CSE Status Register (CSE_SR) . . . 1335
Figure 844. CSE Interrupt Register (CSE_IR) . . . 1337
Figure 845. CSE Error Code Register (CSE_ECR) . . . 1337
Figure 846. CSE Command Register (CSE_CMD) . . . 1338
Figure 847. CSE Parameter Register (CSE_Px) . . . 1339
Figure 848. JTAG STL (IEEE 1149.1) block diagram . . . 1355
Figure 849. 5-Bit Instruction Register . . . 1358
Figure 850. Device Identification Register . . . 1359
Figure 851. CENSOR_CTRL Register . . . 1359
Figure 852. Shifting Data Through a Register . . . 1360
Figure 853. IEEE 1149.1-2001 TAP Controller Finite State Machine . . . 1361
Figure 854. NDI functional block diagram . . . 1366
Figure 855. NDI Implementation block diagram . . . 1367
Figure 856. Multi mode Nexus Tap . . . 1370
Figure 857. Debug Mode Control Interconnections . . . 1375
Figure 858. Nexus Port Controller Block Diagram . . . 1376
Figure 859. 4-Bit Instruction Register . . . 1377
Figure 860. Nexus Device ID Register (DID) . . . 1378
Figure 861. Port Configuration Register (PCR) . . . 1379
Figure 862. MSEO Transfers . . . 1381
Figure 863. Message field sizes . . . 1382
Figure 864. Transmission Sequence of Messages . . . 1383
Figure 865. IEEE 1149.1-2001 TAP Controller State Machine . . . 1384
Figure 866. NEXUS Controller State Machine . . . 1385
Figure 867. IEEE 1149.1 Controller Command Input . . . 1386
Figure 868. Nexus3 Functional Block Diagram . . . 1388
Figure 869. Development Control Register 1 (DC1) . . . 1397
Figure 870. Development Control Register 2 (DC2) . . . 1398
Figure 871. Development Control Register 3 (DC3) . . . 1400

Figure 872. Development Control Register 4 (DC4) 1401
Figure 873. Development Status Register (DS). 1402
Figure 874. Read/Write Access Control/Status Register (RWCS). 1403
Figure 875. Read/Write Access Address Register (RWA). 1404
Figure 876. Read/Write Access Data Register (RWD) 1405
Figure 877. Watchpoint Trigger Register (WT) 1406
Figure 878. Program Trace Start Trigger Control (PTSTC) Register. 1408
Figure 879. Program Trace End Trigger Control (PTETC) Register 1409
Figure 880. Data Trace Start Trigger Control (DTSTC) Register. 1410
Figure 881. Data Trace End Trigger Control (DTETC) Register 1411
Figure 882. Watchpoint Mask Register 1412
Figure 883. Overrun Control Register (OVCR) 1413
Figure 884. Data Trace Control Register (DTC) 1414
Figure 885. Data Trace Start Address Register 1– 4 (DTSA1–DTSA4) 1415
Figure 886. Data Trace End Address Register 1 – 4 (DTEA1–DTEA4). 1415

1 Preface

1.1 Overview

The primary objective of this document is to define the functionality of the SPC564Bxx and SPC56ECxx microcontroller for use by software and hardware developers. The SPC564Bxx and SPC56ECxx 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 SPC564Bxx and SPC56ECxx 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
	– *Microcontroller Boot*	– 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.	

Table 1. Guide to this reference manual (continued)

Chapter		Description	Functional group
#	Title		
15	*e200z0h Core*	Overview on cores. For more details consult the core reference manuals available on www.st.com.	Core platform modules
16	*e200z4d Core*		
17	*Enhanced Direct Memory Access (eDMA)*	Operation and configuration information on the 32-channel direct memory access that can be used to transfer data between any memory mapped locations. Certain peripherals have eDMA triggers that can be used to feed configuration data to, or read results from the peripherals.	
18	*eDMA Channel Multiplexer (DMA_MUX)*	Operation and configuration information for the eDMA multiplexer, which takes the 56 possible eDMA sources (triggers from the DSPI, eMIOS, I^2C, ADC and LINFlexD_) and multiplexes them onto the 32 eDMA channels.	
19	*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.	
20	*Crossbar Switch (XBAR)*	The 8-way Master / Slave crossbar switch can be highly configured to maximize performance based on your application requirements.	
21	*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.	
22	*Semaphores*	Semaphores can be configured as described in this section to ensure memory coherency in multi core microcontrollers.	
23	*Performance Optimization*	Details the configurations that are possible in order to maximize the performance of the cores and system.	Performance
24	*System Integration Unit Lite (SIUL)*	How to configure the pins or ports for input or output functions including external interrupts and DSI serialization.	Ports
25	*Inter-Integrated Circuit Bus Controller Module (I2C)*	These chapters describe the configuration and operation of the various communication modules. Some of these modules support DMA requests to fill / empty buffer queues to minimize CPU overhead.	Communication modules
26	*LIN Controller (LINFlexD)*		
27	*FlexCAN*		
28	*Deserial Serial Peripheral Interface (DSPI)*		
29	*FlexRay Communication Controller (FLEXRAY)*		
30	*Fast Ethernet Controller (FEC)*		

Table 1. Guide to this reference manual (continued)

Chapter		Description	Functional group
#	Title		
31	*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 with Real-Time Interrupt (PIT_RTI)*	Set of 32-bit count down timers that provide periodic events (which can trigger an interrupt) with automatic re-load. The RTI can be used for generating a periodic system wakeup event.	
32	*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
33	*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.	
35	*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
34	*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		
36	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
37	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.	
38	Error Correction Status Module (ECSM)	Provides information about the last reset, general device information, system fault information and detailed ECC error information.	
39	Self-Test Control Unit (STCU)	Details how to configure the Memory Built In Self Test (MBIST), which checks the operational status of the memories at system boot time (according to SIL requirements) and reports accordingly.	
40	Cryptographic Services Engine (CSE)	Implements security features as defined by the SHE specification, providing secure boot authentication and additional AES security features.	
41	JTAG Controller (JTAGC)	Used for boundary scan as well as device debug.	Debug
42	Nexus Development Interface (NDI)	Provides advanced debug features including non intrusive trace capabilities.	
43	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 SPC564Bxx and SPC56ECxx 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*.

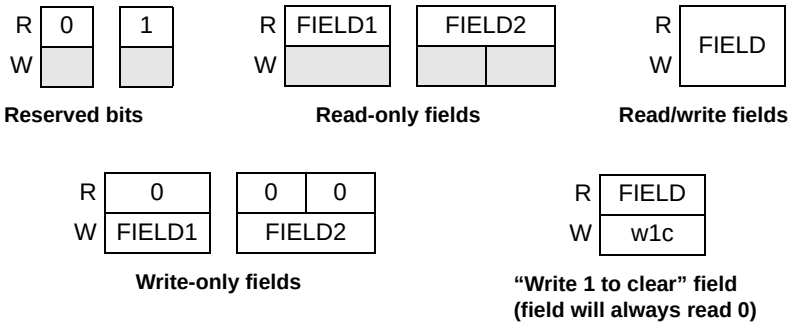


Figure 1. Register figure conventions

The numbering of register bits and fields on SPC564Bxx and SPC56ECxx 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 SPC564Bxx and SPC56ECxx:

- 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 SPC564Bxx and SPC56ECxx documents

This section:

- Describes how the SPC564Bxx and SPC56ECxx documents provide information on the microcontroller
- Makes recommendations on how to use the documents in a system design

1.6.1 The SPC564Bxx and SPC56ECxx document set

The SPC564Bxx and SPC56ECxx 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 SPC564Bxx and SPC56ECxx 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:
  - Configuring CPU memory, branch and cache optimizations
  - Doing detailed software development in assembly language
  - 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 *Chapter 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)	—
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
eDMA Channel Multiplexer	Source values for module eDMA channels	How to connect a module eDMA channel to the eDMA module
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 SPC564Bxx and SPC56ECxx

There are many different approaches to designing a system using the SPC564Bxx and SPC56ECxx so the guidance in this section is provided as an example of how the documents can be applied in this task.

Familiarity with the SPC564Bxx and SPC56ECxx 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 SPC564Bxx and SPC56ECxx requires that certain pins are connected to particular power supplies, system functions and other voltage levels for operation.

The SPC564Bxx and SPC56ECxx 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. *Chapter 4: Signal Description*, describes the power supply pin names, numbers and their purpose. For more detail on the voltage supply of each pin, see *Chapter 11: Voltage Regulators and Power Supplies*. For specifications of the voltage ranges and limits and decoupling of the power supplies see the SPC564Bxx and SPC56ECxx 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 *Chapter 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 *Chapter 4: Signal Description*. The reset pin is bidirectional and its function is closely tied to the reset generation module (*Chapter 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 *Section 6.3: Fast external crystal oscillator (FXOSC) digital interface*, 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. *Chapter 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 *Chapter 24: 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 MMU translates physical memory addresses for use by the CPU and it must be configured before any peripherals or memories are available for use by the CPU. See the *e200z4 Power Architecture Core Reference Manual* for details on how to configure the MMU.

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.

Chapter 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 *Chapter 36: Register Protection*.

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

2 Introduction

2.1 The SPC564Bxx and SPC56ECxx microcontroller family

The SPC564Bxx and SPC56ECxx is a family of Power Architecture® based microcontrollers that target automotive vehicle body and gateway applications such as:

- Central body controller
- Smart junction boxes
- Front modules
- High end gateway
- Combined body controller and gateway

The SPC564Bxx and SPC56ECxx family expands the range of the SPC560B microcontroller family. It provides the scalability needed to implement platform approaches and delivers the performance required by increasingly sophisticated software architectures. The advanced and cost-efficient host processor cores of the SPC564Bxx and SPC56ECxx automotive controller family comply with the Power Architecture embedded category, which is 100 percent user-mode compatible with the original Power Architecture user instruction set architecture (UISA). It operates at speeds of up to 120 MHz and offers high performance processing optimized for low power consumption. It also capitalizes on the nominal available development infrastructure of current Power Architecture devices and is supported with software drivers, operating systems and configuration code to assist with users implementations.

2.2 SPC564Bxx and SPC56ECxx device comparison

Table 3 summarizes the SPC564Bxx and SPC56ECxx family of microcontrollers.

Table 3. SPC564Bxx and SPC56ECxx family comparison[1]

Feature		SPC564B64		SPC56EC64			SPC564B70		SPC56EC70			SPC564B74		SPC56EC74		
Package		**LQFP 176**	**LQFP 208**	**LQFP 176**	**LQFP 208**	**BGA 256**	**LQFP 176**	**LQFP 208**	**LQFP 176**	**LQFP 208**	**BGA256**	**LQFP 176**	**LQFP 208**	**LQFP 176**	**LQFP 208**	**BGA 256**
CPU		e200z4d		e200z4d + e200z0h			e200z4d		e200z4d + e200z0h			e200z4d		e200z4d + e200z0h		
Execution speed[2]		Up to 120 MHz (e200z4d)		Up to 120 MHz (e200z4d) Up to 80 MHz (e200z0h)[3]			Up to 120 MHz (e200z4d)		Up to 120 MHz (e200z4d) Up to 80 MHz (e200z0h)[3]			Up to 120 MHz (e200z4d)		Up to 120 MHz (e200z4d) Up to 80 MHz (e200z0h)[3]		
Code flash memory		1.5 MB					2 MB					3 MB				
Data flash memory		4 x16 KB														
SRAM		128 KB		192 KB			160 KB		256 KB			192 KB		256 KB		
MPU		16-entry														
eDMA[4]		32 ch														
10-bit ADC																
	dedicated[5], [6]	27 ch	33 ch	27 ch	33 ch		27 ch	33 ch	27 ch	33 ch		27 ch	33 ch	27 ch	33 ch	
	shared with 12-bit ADC[7]	19 ch														
12-bit ADC																
	dedicated[8]	5 ch	10 ch	5 ch	10 ch		5 ch	10 ch	5 ch	10 ch		5 ch	10 ch	5 ch	10 ch	
	shared with 10-bit ADC[7]	19 ch														
CTU		64 ch														
Total timer I/O[9] eMIOS		64 ch, 16-bit														
SCI (LINFlexD)		10														
SPI (DSPI)		8														
CAN (FlexCAN)[10]		6														

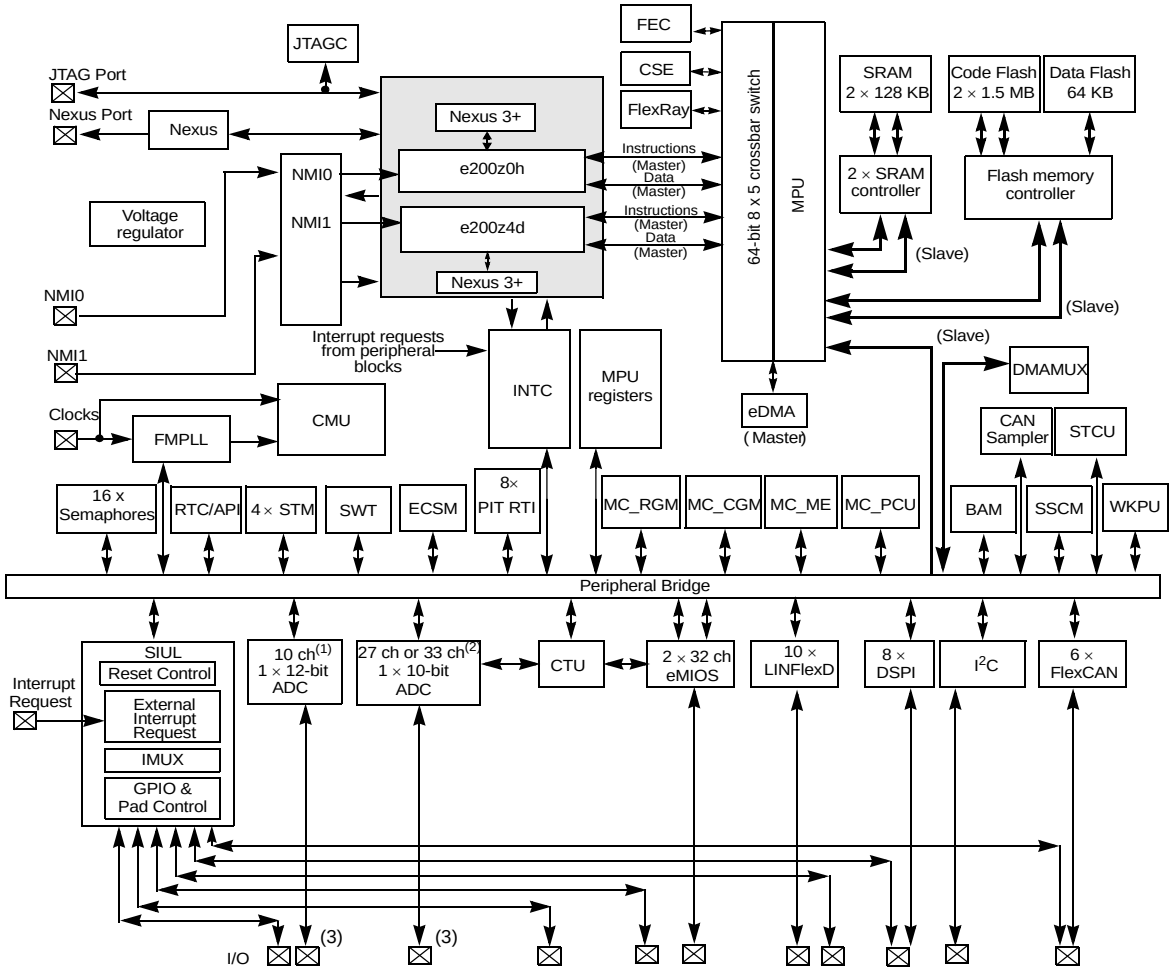
Table 3. SPC564Bxx and SPC56ECxx family comparison[1] (continued)

Feature	SPC564B64		SPC56EC64			SPC564B70		SPC56EC70			SPC564B74		SPC56EC74		
Package	**LQFP 176**	**LQFP 208**	**LQFP 176**	**LQFP 208**	**BGA 256**	**LQFP 176**	**LQFP 208**	**LQFP 176**	**LQFP 208**	**BGA256**	**LQFP 176**	**LQFP 208**	**LQFP 176**	**LQFP 208**	**BGA 256**
FlexRay	Yes														
STCU[11]	Yes														
Ethernet	No		Yes			No		Yes			No		Yes		
I^2C	1														
32 kHz oscillator (SXOSC)	Yes														
GPIO[12]	147	177	147	177	199	147	177	147	177	199	147	177	147	177	199
Debug	JTAG				Nexus 3+	JTAG				Nexus 3+	JTAG				Nexus 3+
Cryptographic Services Engine (CSE)	Optional														

1. Feature set dependent on selected peripheral multiplexing; table shows example.
2. Based on 125 °C ambient operating temperature and subject to full device characterisation.
3. The e200z0h can run at speeds up to 80 MHz. However, if system frequency is >80 MHz (e.g., e200z4d running at 120 MHz) the e200z0h needs to run at 1/2 system frequency. There is a configurable e200z0 system clock divider for this purpose.
4. DMAMUX also included that allows for software selection of 32 out of a possible 57 sources.
5. Not shared with 12-bit ADC, but possibly shared with other alternate functions.
6. There are 23 dedicated ANS plus 4 dedicated ANX channels on LQPF176. For higher pin count packages, there are 29 dedicated ANS plus 4 dedicated ANX channels.
7. 16x precision channels (ANP) and 3x standard (ANS).
8. Not shared with 10-bit ADC, but possibly shared with other alternate functions.
9. As a minimum, all timer channels can function as PWM or Input Capture and Output Control. Refer to the eMIOS section of the device reference manual for information on the channel configuration and functions.
10. CAN Sampler also included that allows ID of CAN message to be captured when in low power mode.
11. STCU controls MBIST activation and reporting.
12. Estimated I/O count for proposed packages based on multiplexing with peripherals.

2.3 Device block diagram

Figure 2 shows a top-level block diagram of the SPC564Bxx and SPC56ECxx.



Legend:

ADC	Analog-to-Digital Converter	JTAGC	JTAG controller
BAM	Boot Assist Module	LINFlexD	Local Interconnect Network Flexible with DMA support
CSE	Cryptographic Services Engine	MC_ME	Mode Entry Module
CAN	Controller Area Network (FlexCAN)	MC_CGM	Clock Generation Module
CMU	Clock Monitor Unit	MC_PCU	Power Control Unit
CTU	Cross Triggering Unit	MC_RGM	Reset Generation Module
DMAMUX	DMA Channel Multiplexer	MPU	Memory Protection Unit
DSPI	Deserial Serial Peripheral Interface	Nexus	Nexus Development Interface
eDMA	enhanced Direct Memory Access	NMI	Non-Maskable Interrupt
FlexCAN	Controller Area Network controller modules	PIT_RTI	Periodic Interrupt Timer with Real-Time Interrupt
FEC	Fast Ethenet Controller	RTC/API	Real-Time Clock/ Autonomous Periodic Interrupt
eMIOS	Enhanced Modular Input Output System	SIUL	System Integration Unit Lite
ECSM	Error Correction Status Module	SRAM	Static Random-Access Memory
FMPLL	Frequency-Modulated Phase-Locked Loop	SSCM	System Status Configuration Module
FlexRay	FlexRay Communication Controller	STM	System Timer Module
I2C	Inter-integrated Circuit Bus	SWT	Software Watchdog Timer
IMUX	Internal Multiplexer	STCU	Self Test Control Unit
INTC	Interrupt Controller	WKPU	Wakeup Unit

Notes:
1) 10 dedicated channels plus up to 19 shared channels. See the device-comparison table.
2) Package dependent. 27 or 33 dedicated channels plus up to 19 shared channels. See the device-comparison table.
3) 16 x precision channels (ANP) are mapped on input only I/O cells.

Figure 2. SPC564Bxx and SPC56ECxx block diagram

2.4 Feature summary

2.4.1 High-performance e200z4d core processor

The e200z4d core includes the following features:

- Dual issue, 32-bit Power Architecture 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 target prefetching using 8-entry BTB
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and flash memory via independent Instruction and Data BIUs
- Load/store unit
  - 2 cycle load latency
  - Fully pipelined
  - Big- and little-endian support
  - Misaligned access support
- 64-bit General Purpose Register file
- Dual AHB 2.v6 64-bit system buses
- Memory Management Unit (MMU) with 16-entry fully-associative translation lookaside buffer (TLB) and multiple page size support
- 4 KB, 2/4-way set associative instruction cache
- Embedded Floating-Point APU (EFPU) supporting scalar single-precision floating-point operations
- Signal Processing Extension (SPE1.1) APU supporting SIMD fixed-point operations using the 64-bit General Purpose Register file.
- Embedded Floating-Point (EFP2) APU supporting scalar and vector SIMD single-precision floating-point operations, using the 64-bit General Purpose Register file.
- Nexus Class 3+ real-time development unit
- Power management
  - Low power design–extensive clock gating
  - Power saving modes: nap, sleep, wait
  - Dynamic power management of execution units, cache and MMU

2.4.2 e200z0h core processor

The e200z0h core includes the following features:

- High performance, low-cost e200z0h core processor for managing peripherals and interrupts
- Single issue 4-stage pipelined in-order execution, 32-bit Power Architecture CPU
- Variable length encoding (VLE), allowing mixed 16-bit and 32-bit instructions
  - Results in efficient code size footprint
  - Minimizes impact on performance
- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1-cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- 32-bit general purpose registers (GPRs)
- Separate instruction bus and load/store bus Harvard architecture
- Hardware vectored interrupt support
- Reservation instructions for implementing read-modify-write constructs
- Multi-cycle divide word (divw) and load multiple word (lmw) store multiple word (smw) multiple class instructions, can be interrupted to prevent increases in interrupt latency
- Extensive system development support through Nexus 3 debug port

2.4.3 Memory Built-In Self Test (MBIST)

- User selectable MBIST that can be enabled to run out of various reset conditions. User MBIST can also be disabled.
- Configurable fault response (Critical fault and non-critical fault)

2.4.4 Enhanced Direct Memory Access Controller (eDMA)

The following summarizes the SPC564Bxx and SPC56ECxx implementation of the eDMA controller:

- 32 channels support independent 8, 16 or 32-bit single value or block transfers
- Supports variable sized queues and circular queues
- Source and destination address registers are independently configured to post-increment or remain constant
- Each transfer is initiated by a peripheral, CPU, periodic timer interrupt or eDMA channel request
- Peripheral DMA request sources possible from DSPI's, I^2C, 10-bit ADC, 12-bit ADC, eMIOS, GPIOs and LINFlexD.
- Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer
- DMA transfers is possible between system memories and all accessible memory mapped locations including peripheral and registers.
- Programmable DMA Channel Mux allows assignment of any DMA source to any available DMA channel with up to a total of 64 potential request sources.

2.4.5 Error Correction Status Module (ECSM)

The ECSM on this device manages the ECC configuration and reporting for the platform memories (flash memory and SRAM). It does not implement the actual ECC calculation. The following errors and indications are reported into the ECSM dedicated registers:

- ECC error status and configuration for flash memory and SRAM
- ECC error reporting for flash memory
- ECC error reporting for SRAM
- ECC error injection for SRAM

2.4.6 Crossbar Switch (XBAR)

The following summarizes the SPC564Bxx and SPC56ECxx's implementation of the crossbar switch:

- Eight master ports
  - Masters: e200z0h and e200z4d instruction buses, e200z0h and e200z4d data buses, eDMA, FlexRay, Ethernet and CSE
- Multiple bus slaves to enable access to flash memory, SRAM, PBridge
- Fully concurrent transfers between independent master and slave port
- Fixed priority and round robin arbitration independently programmable for each slave

2.4.7 Memory Protection Unit (MPU)

The following list summarizes the MPU features:

- 16 region descriptors for per-master protection
- Start and end address defined with 32-byte granularity
- Overlapping regions supported
- Protection attributes can optionally include process ID
- Protection offered for 5 concurrent read ports
- Read and write attributes for all masters
- Execute and supervisor/user mode attributes for processor masters

2.4.8 Interrupt Controller (INTC)

The SPC564Bxx and SPC56ECxx implements an interrupt controller that features the following:

- Unique 9-bit vector for each of the 246 separate interrupt sources
- Dual core interrupt controller
  - Each interrupt can be targeted to e200z4d, e200z0h or both
- 8 software triggerable interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Ability to modify the ISR or task priority.
  - Modifying the priority can be used to implement the Priority Ceiling Protocol for accessing shared resources.
- External high priority interrupts directly accessing the e200z0h core and e200z4d core critical interrupt mechanism

2.4.9 System clocks and clock generation

The following list summarizes the system clock and clock generation on the SPC564Bxx and SPC56ECxx:

- System clock can be derived from the following sources
  - External crystal oscillator (4–40 MHz)
  - FMPLL
  - 16 MHz internal RC oscillator
- Programmable divider for output clock
- Separate programmable peripheral bus clock dividers for each peripheral domain
- Frequency Modulated Phase-Locked Loop (FMPLL)
  - Input clock frequency from 4 MHz to 40 MHz
  - Selectable clock source from external oscillator or internal 16 MHz RC oscillator
  - Lock detect circuitry continuously monitors lock status
  - Loss of clock (LOC) detection for reference and feedback clocks
  - On-chip loop filter (for improved electromagnetic interference performance and reduces number of external components required)
  - Auxiliary output that can be used to clock FlexRay
  - Progressive clock switching to reduce current surge at FMPLL startup
- On-chip crystal oscillator supporting 4 MHz to 40 MHz crystals.
- Dedicated 16 MHz internal RC oscillator
  - Used as default clock source out of reset
  - Provides a clock for rapid start-up from low power modes
  - Provides a back-up clock in the event of PLL or External oscillator clock failure
  - 5% accuracy over the operating temperature range
  - Trimming registers to support frequency adjustment with in-application calibration facilitated by the CMU
- Dedicated internal 128 kHz internal RC oscillator for low power mode operation and self wake-up
  - 5% accuracy
  - Trimming registers to support improved accuracy with in-application calibration facilitated by the CMU
- 32 kHz low power external oscillator for accurate low-power real time clock

2.4.10 System Integration Unit Lite (SIUL)

The SIUL features the following:

- Up to five levels of internal pin muliplexing, allowing exceptional flexibility in the allocation of device functions for each package
- Centralized general purpose input output (GPIO) control of up to 199 input/output pins (package dependent)
- All GPIO pins can be independently configured to support pull-up, pull down, or no pull.
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- All peripheral pins, with the exception of precision ADC channels, can be configured as either general purpose input or output pins. Precision ADC channels can only be configured as general purpose inputs.
- Direct readback of the pin value supported on all digital output pins through the SIU
- Configurable digital input filter that can be applied to up to 24 general purpose input pins for noise elimination on external interrupts.
- Register configuration protected against change with soft lock for temporary guard or hard lock to prevent modification until next reset.
- Support for 4 parallel input select muxes

2.4.11 Software Watchdog Timer (SWT)

The SWT on the SPC564Bxx and SPC56ECxx features the following:

- 32-bit time-out register to set the time-out period
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

2.4.12 Flash memory

The on-chip flash memory on the SPC564Bxx and SPC56ECxx features the following:

- 3 MB burst flash memory
- Single dual port PFlash controller and Flash BIU shared with the data flash memory
- Flash memory partitioning:
  - 1.5 MB code flash memory module 1
    - 1 × 512 KB (2 × 16 KB, 3 × 32 KB, 3 × 128 KB, 2 × 16 KB (reserved))
    - 2 × 512 KB (4 × 128 KB)
  - 1.5 MB code flash memory module 2
    - 1 × 512 KB (2 × 16 KB, 3 × 32 KB, 3 × 128 KB, 2 × 16 KB (reserved))
    - 2 × 512 KB (4 × 128 KB)
  - 64 KB data flash memory
    - 4 × 16 KB, 1 × 8 KB (reserved)
- RWW is supported between both the 1.5M code flash memory and data flash memory modules, to facilitate the EEPROM emulation capability. RWW is not supported between the 512 KB arrays within the 1.5M code flash memory.

- Typical code flash memory access time: 40 ns
  - 0 wait-state for buffer hits
  - 5 additional wait-states for page buffer miss at 120+2% MHz
- Typical data flash memory access time is 120 ns: up to 13 wait-states for page buffer miss at 120+2% MHz.
- Page buffers can be allocated for code-only, fixed partitions of code and data, all available for any access
- 64-bit ECC with single-bit correction, double-bit detection for data integrity in code flash memory
- 32-bit ECC with single-bit correction, double-bit detection for data integrity in data flash memory
- Censorship protection scheme to prevent flash content visibility
- Supports flash writes using internal 16 MHz RC oscillator
- Margin read for flash array supported for initial program verification
- Flash memory partitioning as shown in *Table 4*

Table 4. SPC564Bxx and SPC56ECxx flash memory partitioning

Array	Address	SPC564B64 / SPC56EC6	SPC564B70 / SPC56EC70	SPC564B74 / SPC56EC4
		1.5 MB	2 MB	3 MB
Array_A	Flash Base + 0x0000_0000	32 KB	32 KB	32 KB
	Flash Base + 0x0000_8000	16 KB	16 KB	16 KB
	Flash Base + 0x0000_C000	16 KB	16 KB	16 KB
	Flash Base + 0x0001_0000	32 KB	32 KB	32 KB
	Flash Base + 0x0001_8000	32 KB	32 KB	32 KB
	Flash Base + 0x0002_0000	128 KB	128 KB	128 KB
	Flash Base + 0x0004_0000	128 KB	128 KB	128 KB
	Flash Base + 0x0006_0000	128 KB	128 KB	128 KB
Array_B	Flash Base + 0x0008_0000	128 KB	128 KB	128 KB
	Flash Base + 0x000A_0000	128 KB	128 KB	128 KB
	Flash Base + 0x000C_0000	128 KB	128 KB	128 KB
	Flash Base + 0x000E_0000	128 KB	128 KB	128 KB

Table 4. SPC564Bxx and SPC56ECxx flash memory partitioning (continued)

Array	Address	SPC564B64 / SPC56EC6	SPC564B70 / SPC56EC70	SPC564B74 / SPC56EC4
		1.5 MB	2 MB	3 MB
Array_C	Flash Block + 0x0010_0000	128 KB	128 KB	128 KB
	Flash Base + 0x0012_0000	128 KB	128 KB	128 KB
	Flash Base + 0x0014_0000	128 KB	128 KB	128 KB
	Flash Base + 0x0016_0000	128 KB	128 KB	128 KB
Array_D	Flash Base + 0x0018_0000	—	128 KB	128 KB
	Flash Base + 0x001A_0000	—	128 KB	128 KB
	Flash Base + 0x001C_0000	—	128 KB	128 KB
	Flash Base + 0x001E_0000	—	128 KB	128 KB
Array_E	Flash Base + 0x0020_0000	—	—	128 KB
	Flash Base + 0x0022_0000	—	—	128 KB
	Flash Base + 0x0024_0000	—	—	128 KB
	Flash Base + 0x0026_0000	—	—	128 KB
Array_F	Flash Base + 0x0028_0000	—	—	32 KB
	Flash Base + 0x0028_8000	—	—	16 KB
	Flash Base + 0x0028_C000	—	—	16 KB
	Flash Base + 0x0029_0000	—	—	32 KB
	Flash Base + 0x0029_8000	—	—	32 KB
	Flash Base + 0x002A_0000	—	—	128 KB
	Flash Base + 0x002C_0000	—	—	128 KB
	Flash Base + 0x002E_0000	—	—	128 KB

Table 4. SPC564Bxx and SPC56ECxx flash memory partitioning (continued)

Array	Address	SPC564B64	SPC56EC6	SPC564B70	SPC56EC70	SPC564B74	SPC56EC4
		1.5 MB		2 MB		3 MB	
Array_G	Data Flash Base + 0x0000_0000	16 KB		16 KB		16 KB	
	Data Flash Base + 0x0000_4000	16 KB		16 KB		16 KB	
	Data Flash Base + 0x0000_8000	16 KB		16 KB		16 KB	
	Data Flash Base + 0x0000_C000	16 KB		16 KB		16 KB	

2.4.13 On-chip SRAM

On-chip SRAM on the SPC564Bxx and SPC56ECxx features the following:

- 256 KB in total
  - 2 separate 128 KB blocks on different slave ports (for maximum performance)
- 8 KB, 40 KB, 64 KB or 96 KB of RAM can be retained in standby mode
- 64-bit RAM organization with ECC, and redundancy
- Available for data and program
- 0 wait state for 64-bit accesses (64-bit aligned) up to 64 MHz, after which 1 wait state is compulsory for operations above 64 MHz.
- Typical SRAM access time at less than 64 MHz: 0 wait-state for reads and 32-bit writes; 1 wait-state for 8- and 16-bit writes if back to back with a read to same memory block. Above 64 MHz + 4%, additional RAM wait state needs to be added.
- 32-bit ECC with single-bit correction, double bit detection for data integrity
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory. User transparent ECC encoding and decoding for byte, half word, and word accesses

2.4.14 Boot Assist Module (BAM)

The SPC564Bxx and SPC56ECxx BAM is implemented as follows:

- Block of read-only memory containing VLE code which is executed according to boot mode of the device
- Download of code into internal SRAM possible via FlexCAN or LINFlexD, after which code can be executed

2.4.15 System Status and Configuration Module (SSCM)

The SSCM includes these distinctive features:

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

2.4.16 Enhanced Modular Input Output System (eMIOS)

The SPC564Bxx and SPC56ECxx implements a scaled-down version of the eMIOS module:

- Up to 64 (2 x 32 ch) timed I/O channels with 16-bit counter resolution
- Buffered updates
- Support for shifted PWM outputs to minimize occurrence of concurrent edges
- Supports configurable trigger outputs for ADC conversion for synchronization to channel output waveforms
- Edge-aligned output pulse width modulation
  - Programmable pulse period and duty cycle
  - Supports 0% and 100% duty cycle
  - Shared or independent time bases
- DMA transfer support available

Table 5 shows the supported eMIOS modes.

Table 5. Supported eMIOS channel modes

Mode		Channel type			
Description	**Name**	**Counter / OPWM / ICOC**	**O(I)PWM / OPWFMB / OPWMCB / ICOC**	**O(I)PWM / ICOC**	**OPWM / ICOC**
Double action output compare	DAOC	x	x	x	—
General purpose input / output	GPIO	x	x	x	x
Input filter	IPF	x	x	x	x
Input period measurement	IPM	x	x	x	—
Input pulse width measurement	IPWM	x	x	x	—

a. When MBIST is enabled, SSCM reads STCU self test parameters from NVM and copies into STCU. (See Section 35.2.5.27, “Nonvolatile User Options register (NVUSRO)).
When CSE is supported, the SSCM logic issues the SECURE_BOOT command to the CSE which starts the secure boot process.

Table 5. Supported eMIOS channel modes (continued)

Mode		Channel type			
Description	**Name**	**Counter / OPWM / ICOC**	**O(I)PWM / OPWFMB / OPWMCB / ICOC**	**O(I)PWM / ICOC**	**OPWM / ICOC**
Modulus counter	MC	x	—	—	—
Modulus counter buffered (up / down)	MCB	x	x	—	—
Output pulse width and frequency modulation buffered	OPWFMB	x	x	—	—
Output pulse width modulation buffered	OPWMB	—	x	x	x
Center aligned output PWM buffered with dead time	OPWMCB	—	x	—	—
Output pulse width modulation trigger	OPWMT	x	x	x	x
Pulse edge accumulation	PEA	x	—	—	—
Pulse edge counting	PEC	x	—	—	—
Quadrature decode	QDEC	x	—	—	—
Single action input capture	SAIC	x	x	x	x
Single action output compare	SAOC	x	x	x	x

Table 6 shows the maximum eMIOS channel allocation.

Table 6. eMIOS configuration

Channel type	Maximum number of channels		Total
	eMIOS_0	**eMIOS_1**	
Counter / OPWM / ICOC(1)	5	5	10
O(I)PWM / OPWFMB / OPWMCB / ICOC(2)	7	0	7
O(I)PWM / ICOC(3)	7	7	14
OPWM / ICOC(4)	13	20	33

1. Each channel supports a range of modes including Modulus counters, PWM generation, Input Capture, Output Compare.
2. Each channel supports a range of modes including PWM generation with dead time, Input Capture, Output Compare.
3. Each channel supports a range of modes including PWM generation, Input Capture, Output Compare, Period and Pulse width measurement.
4. Each channel supports a range of modes including PWM generation, Input Capture, Output Compare.

2.4.17 Analog-to-Digital Converter (ADC)

The ADC features the following:

- 2 ADC modules, one 10-bit resolution (ADC0) and one 12-bit resolution (ADC1) supporting synchronous conversions on channels
- 0–V_{DD} common mode conversion range
- Independent reference supplies for each ADC
- Conversions times of < 1 µs available
- Up to 19 shared channels, of which 16 are high-precision on dedicated pins (called ANP) and are not multiplexed with any other function in order to improve the accuracy. All other channels are multiplexed with other functions.
  - 19 channels shared between 10-bit and 12-bit ADC
  - 10 channel 12-bit ADC
  - Up to 29 channel 10-bit ADC (for 176-pin LQFP) or up to 33 channel 10-bit ADC (208-pin LQFP, BGA208 and BGA256)
- Externally multiplexed channels (ANX)
  - Internal control to support generation of external analog multiplexor selection
  - 4 internal channels optionally used to support externally multiplex inputs, providing transparent control for additional ADC channels
  - Each of the 4 channels supports up to 8 externally muxed inputs
  - Individual dedicated result register also available for externally muxed conversion channels
  - 3 independently configurable sample and conversion times for high occurrence channels, internally muxed channels and externally muxed channels
- Configurable right-aligned or left-aligned result formats
- Support for one-shot, scan and injection conversion modes
- Independently configurable parameters for channels:
  - Offset refresh
  - Sampling
- Conversion triggering support
  - Internal conversion triggering from PIT_RTI or timed I/O module (eMIOS) through cross triggering unit (CTU)
- Up to 6 (for ADC0) and up to 3 (for ADC1) configurable analog comparator channels offering range comparison with triggered alarm
  - Greater than
  - Less than
  - Out of range
- All unused analog pins (16 ANP) available as general purpose input pins
- Unused 10-bit ADC analog pins, with the with exception of the 16 ANP shared channels available as general purpose input/output pins
- Power-down mode
- Supports DMA transfer of results based on end of conversion chain or each conversion

2.4.18 Cross Triggering Unit (CTU)

The CTU enables the synchronization of ADC conversions with a timer event. Its key features are:

- Single cycle delayed trigger output; trigger output is a combination of 64 (generic value) input flags/events connected to different timers in the system
- Triggers ADC conversions from any eMIOS channel
- Triggers ADC conversions from up to 2 dedicated PIT_RTIs
- Maskable interrupt generation whenever a trigger output is generated
- 1 event configuration register dedicated to each timer event allows to define the corresponding ADC channel
- Acknowledgment signal to eMIOS/PIT_RTI for clearing the flag
- Synchronization with ADC to avoid collision

2.4.19 Deserial Serial Peripheral Interface (DSPI)

The DSPI features the following:

- 8 DSPI modules supported
- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable baud rate
- Programmable data frames from 4 to 32-bits
- Support for serial chaining of several DSPI modules
- Pin serialization with interleaved SPI frames for control and diagnostics
- Continuous serial communications clock
- Parity control
- Support for the downstream Micro Second Channel with Timed Serial Bus (TSB) configuration.
- Up to 6 chip select lines available, depending on package and pin multiplexing, enable 64 external devices to be selected using external muxing from a single DSPI
- Up to 8 independently configurable transfer types can be configured for each DSPI using the clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for de-glitching
- FIFOs for buffering up to 4 transfers on the transmit and receive side
- General purpose I/O functionality on pins when not used for SPI
- Queueing operation possible through use of eDMA
- DSI mode (allows the serialization of eMIOS waveforms)

2.4.20 Serial communication interface (LINFlexD)

The LINFlexD features the following:

- 10 LINFlexD modules supported
- Supports LIN Master mode, LIN Slave mode and UART mode
- One module supporting LIN Master and Slave mode, 9 modules supporting LIN Master only mode
- LIN state machine compliant to LIN1.3, 2.0 and 2.1 Specifications
- Handles LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store Identified and up to eight data bytes
  - Supports message length of up to 64bytes
  - Detection and flagging of LIN errors
    - Sync field; Delimiter; ID parity; Bit, Framing; Checksum and Timeout errors
  - Classic or extended checksum calculation
  - Configurable break duration of up to 50 bit times
  - Programmable Baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features
    - Loop back; Self Test; LIN bus stuck dominant detection
  - Interrupt driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling
  - 16 ID filters for discarding of irrelevant LIN responses
- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-bytes receive, 4-bytes transmit
  - Configurable word length (8-bit, 9-bit, 16-bit or 17-bit words)
  - Error detection and flagging
    - Parity, Noise and Framing errors
  - Interrupt driven operation with four interrupts sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud-rate modulus counter and 16-bit fractional
  - Two receiver wake-up methods
- Support for DMA enabled transfers on all LIN modules

2.4.21 Controller Area Network (FlexCAN)

The enhanced FlexCAN module features the following:

- 6 FlexCAN modules supported
- Compliant with CAN protocol specification, Version 2.0B active
- 64 mailboxes, each configurable as transmit or receive
  - Mailboxes configurable while module remains syncronised to CAN bus
- Transmit features
  - Arbitration scheme according to message ID or message buffer number and local priority
  - Internal arbitration to guarantee no inner priority inversion
  - Multiple transmit buffers to avoid outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - Hardware FIFO can be enabled
  - 8 mailboxes can be configured to provide a 6-entry receive FIFO and 8 programmable acceptance filters
- Programmable clock source
  - system clock
  - Direct oscillator clock to avoid PLL jitter
- Listen only mode capabilities
- CAN Sampler available to be connected to one of the available CAN module pads
  - supports capturing of first message's identifier while in STOP mode

2.4.22 Fast Ethernet Controller (FEC)

The FEC incorporates the following features

- Support for 3 different physical interfaces
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 10-Mbps 7-wire interface (industry standard)
- Built in FIFO and DMA controller
- IEEE 802.3 MAC (compliant with IEEE 802.3 1998 edition)
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- IEEE 802.3 full duplex flow control
- Support for full duplex operation (200 Mbps throughput) with a system clock of 100 MHz using the external TX_CLK or RX_CLK
- Support for full duplex operation(100 Mbps throughput) with a system clock of 50 MHz using the external TX_CLK or RX_CLK
- Retransmission from transmit FIFO following a collision (no system bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no system bus utilization)
- Address recognition
  - Frames with broadcast address may be always accepted or always rejected
  - Exact match for single 48-bit individual (unicast) address
  - Hash (64-bit hash) check of individual (unicast) addresses
  - Hash (64-bit hash) check of group (multicast) addresses
  - Promiscuous mode
- RMON and IEEE statistics
- Interrupts for network activity and error conditions

2.4.23 Cryptographic Services Engine (CSE)

The CSE is a cryptographic module which supports the encoding and decoding of any kind of data. The module fulfills all requirements of the SHE specification.

Potential use-cases of the CSE module are:

- Secure Boot
- Software Update, Software and “Know-How” protection
- Prevent “chip tuning”
- Immobilizer
- Component Protection
- Secure Network
- Digital Right Management

The CSE has the following features:

- Secure storage for cryptographic keys
- AES-128 encryption and decryption
- AES-128 CMAC authentication
- Random number generation
- Secure boot mode
- System bus master interface

2.4.24 Dual-Channel FlexRay Controller

The FlexRay controller provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A* compliant bus driver interface
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5Mbit/s and 2.5 Mbit/s supported.
- Up to 128 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory
  - Allows for flexible and efficient message buffer implementation
  - Consistent data access ensured by means of buffer locking scheme
  - Application can lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0 up to 254 bytes
- Two independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
  - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as:
  - Receive message buffer
  - Single buffered transmit message buffer
  - Double buffered transmit message buffer (combines two single buffered message buffers.
- Individual message buffer reconfiguration supported
  - means provided to safely disable individual message buffers
  - disabled message buffers can be reconfigured
- Two independent receive FIFOs
  - One receive FIFO per channel
  - Up to 255 entries for each FIFO
  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for dynamic segment
- Four configurable slot error counters
- Four dedicated slot status indicators

  - Used to observe slots without using receive message buffers
- Measured value indicators for the clock synchronization
  - Internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay memory
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- One absolute timer
- One timer that can be configured to absolute or relative
- SECDED for protocol engine data RAM
- SEDDED for chi lookup table RAM
- ECC supported on internal RAMs
- Can be clocked from external 40 MHz crystal or on-chip PLL

2.4.25 Inter-IC Communications (I^2C) module

The I^2C module features the following:

- One I^2C module supported
- Two-wire bi-directional serial bus for on-board communications
- Compatibility with I^2C bus standard
- Multimaster 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

2.4.26 Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI)

The PIT_RTI features the following:

- General purpose interrupt timers
- Interrupt timers for triggering ADC conversions
- Interrupt timers for triggering DMA transfers
- 32-bit counter resolution
- RTI support
- Clocked by system clock frequency
- Real Time Interrupt (RTI)
  - RTI can be clocked by 4-40 MHz crystal

2.4.27 System Timer Module (STM)

One STM supports the following:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels running off the same up-counter
- Independent interrupt source for each channel
- Clocked by the main system clock
- Counter can be stopped in debug mode

2.4.28 Real Time Counter/ Autonomous Periodic Interrupt (RTC/API)

The RTC/API features the following:

- Local 512 and 32 clock dividers
- Configurable resolution for different timeout periods
  - 1 s resolution for > 1 hour period
  - 1 ms resolution for 2 second period
- Selectable clock sources (all sources pass through clock dividers)
  - 32 kHz slow external crystal oscillator
  - 128 kHz slow internal RC oscillator
  - 16 MHz fast internal RC oscillator
  - High frequency crystal oscillator supporting external crystals in the range of 4 MHz to 40 MHz

2.4.29 Nexus Development Interface (NDI)

Nexus features the following:

- 21-bit full duplex pin interface for high throughput, including existing four JTAG pins
  - Two modes are supported: full port mode (FPM) and reduced port mode (RPM). FPM comprises 12 MDO pins. RPM comprises 8 MDO pins, and can be used to increase the number of GPIOs. Care must be taken as bandwidth will be limited in RPM.
  - Auxiliary output port
    - One MCKO (Message clock out) pin
    - 12 MDO (Message data out) pins
    - One $\overline{\text{MSEO}}$ (Message start/end out) pins
    - One $\overline{\text{EVTO}}$ (Event out) pin
  - Auxiliary input port uses one $\overline{\text{EVTI}}$ (Event in) pin
  - JTAG port uses four pins (TDI, TDO, TMS, and TCK)
- The NPC block performs the following functions:
  - Controls arbitration for ownership of the Nexus Auxiliary Output Port between e200z0h and e200z4d Nexus
  - Nexus Device Identification Register and Messaging
  - Generates MCKO enable and frequency division control signals
  - Controls sharing of EVTO between e200z0h and e200z4d cores.

  - Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle.
  - Control of the device-wide debug mode
  - Generates asynchronous reset signal for Nexus blocks
- Host processor (e200z4d) and secondary processor (e200z0h) development support features:
  - IEEE-ISTO 5001-2010 standard class 3+ compliant.
    - Program trace via Branch Trace Messaging (BTM), with the option of using Branch history messaging to enhance message throughput.
    - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
    - 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 development tools to trace ownership flow.
    - Run-time access to the on-chip memory map via the JTAG port.
    - Watchpoint messaging (WPM) via the auxiliary port. This allows a watchpoint to be set, which then sends a watchpoint message each time the watchpoint is hit. Unlike watchpoint triggering, WPM does not stop the core or start trace.
    - Registers for Program Trace, Data Trace, Ownership Trace and Watchpoint Trigger.
  - Additional class 4 features
    - Watchpoint trigger enable of program and/or data trace messaging. This is an extension of WPM to allow a watchpoint to stop or start program or data trace.
    - Processor overrun control
  - All features controllable and configurable via JTAG port.
- Cross triggering — The capability for an EVTO (event out) signal from either the e200z4d or e200z0h Nexus3+ to generate a debug request to the other core, thus allowing both cores to enter debug mode within a short period of each other.

2.4.30 JTAG controller (JTAGC)

JTAG features the following:

- JTAG low pin count interface (IEEE 1149.1) test access port (TAP) interface
- Backward compatible to standard JTAG IEEE 1149.1-2001 test access port (TAP) interface
- Supporting boundary scan testing

2.4.31 On-chip voltage regulator (VREG)

The on-chip voltage regulator includes the following features:

- Support for external ballast transistor
- Regulates 3V to 5V input to generate all internal supplies for internal control
- Manages power gating
- Low power regulators supports operation when in STOP and STANDBY modes to minimize power consumption
- Fast startup on-chip regulators for rapid exit of low power modes
- Low voltage reset supported on all internal supplies

3 Memory Map

Table 7 shows the memory map for the SPC564Bxx and SPC56ECxx. 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 7. SPC564Bxx and SPC56ECxx memory map

Start address	End address	Size (KB)	Region name
0x0000_0000	0x0000_7FFF	32	Code flash memory 0 array 0
0x0000_8000	0x0000_BFFF	16	
0x0000_C000	0x0000_FFFF	16	
0x0001_0000	0x0001_7FFF	32	
0x0001_8000	0x0001_FFFF	32	
0x0002_0000	0x0003_FFFF	128	
0x0004_0000	0x0005_FFFF	128	
0x0006_0000	0x0007_FFFF	128	
0x0008_0000	0x0009_FFFF	128	Code flash memory 0 array 1
0x000A_0000	0x000B_FFFF	128	
0x000C_0000	0x000D_FFFF	128	
0x000E_0000	0x000F_FFFF	128	
0x0010_0000	0x0011_FFFF	128	Code flash memory 0 array 2
0x0012_0000	0x0013_FFFF	128	
0x0014_0000	0x0015_FFFF	128	
0x0016_0000	0x017_FFFF	128	
0x0018_0000	0x0019_FFFF	128	Code flash memory 1 array 2
0x001A_0000	0x001B_FFFF	128	
0x001C_0000	0x001D_FFFF	128	
0x001E_0000	0x001F_FFFF	128	
0x0020_0000	0x0021_FFFF	128	Code flash memory 1 array 1
0x0022_0000	0x0023_FFFF	128	
0x0024_0000	0x0025_FFFF	128	
0x0026_0000	0x0027_FFFF	128	

Table 7. SPC564Bxx and SPC56ECxx memory map (continued)

Start address	End address	Size (KB)	Region name
0x0028_0000	0x0028_7FFF	32	Code flash memory 1 array 0
0x0028_8000	0x0028_BFFF	16	
0x0028_C000	0x0028_FFFF	16	
0x0029_0000	0x0029_7FFF	32	
0x0029_8000	0x0029_FFFF	32	
0x002A_0000	0x002B_FFFF	128	
0x002C_0000	0x002D_FFFF	128	
0x002E_0000	0x002F_FFFF	128	
0x0030_0000	0x007F_FFFF	5120	Reserved
0x0080_0000	0x0080_3FFF	16	Data flash memory array 0
0x0080_4000	0x0080_7FFF	16	
0x0080_8000	0x0080_BFFF	16	
0x0080_C000	0x0080_FFFF	16	
0x0081_0000	0x00E0_7FFF	10214	Reserved
0x00E0_8000	0x00E0_BFFF	16	Code flash memory array 1 test sector
0x00FF_C000	0x00FF_FFFF	16	Code flash memory array 0 shadow sector
0x0100_0000	0x1FFF_FFFF	507,904	Flash memory emulation mapping
0x2000_0000	0x3FFF_FFFF	524,288	Reserved
0x4000_0000	0x4001_FFFF	128	SRAM_1
0x4002_0000	0x4003_FFFF	128	SRAM_2
0x4004_0000	0xBFFF_FFFF	2,096,960	Reserved
Off-platform peripherals (PBRIDGE)			
0xC000_0000	0xC3F8_7FFF	65,056	Reserved
0xC3F8_8000	0xC3F8_BFFF	16	Code flash memory 0 configuration
0xC3F8_C000	0xC3F8_FFFF	16	Data flash memory 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
0xC3FA_4000	0xC3FA_7FFF	16	eMIOS_1
0xC3FA_8000	0xC3FA_FFFF	32	Reserved
0xC3FB_0000	0xC3FB_3FFF	16	Code flash memory 1 configuration
0xC3FB_4000	0xC3FD_7FFF	144	Reserved
0xC3FD_8000	0xC3FD_BFFF	16	SSCM

Table 7. SPC564Bxx and SPC56ECxx memory map (continued)

Start address	End address	Size (KB)	Region name
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_RTI
0xC3FF_4000	0xC3FF_7FFF	16	STCU
0xC3FF_8000	0xFFDF_FFFF	981,024	Reserved
0xFFE0_0000	0xFFE0_3FFF	16	ADC_0
0xFFE0_4000	0xFFE0_7FFF	16	ADC_1
0xFFE0_C000	0xFFE2_FFFF	176	Reserved
0xFFE3_0000	0xFFE3_3FFF	16	I2C
0xFFE3_4000	0xFFE3_FFFF	48	Reserved
0xFFE4_0000	0xFFE4_3FFF	16	LINFlexD_0
0xFFE4_4000	0xFFE4_7FFF	16	LINFlexD_1
0xFFE4_8000	0xFFE4_BFFF	16	LINFlexD_2
0xFFE4_C000	0xFFE4_FFFF	16	LINFlexD_3
0xFFE5_0000	0xFFE5_3FFF	16	LINFlexD_4
0xFFE5_4000	0xFFE5_7FFF	16	LINFlexD_5
0xFFE5_8000	0xFFE5_BFFF	16	LINFlexD_6
0xFFE5_C000	0xFFE5_FFFF	16	LINFlexD_7
0xFFE6_0000	0xFFE6_3FFF	16	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 0xC3F8_0000 — 0xC3FF_FFFF
0xFFF0_0000	0xFFF0_3FFF	16	Reserved
0xFFF0_4000	0xFFF0_7FFF	16	XBAR
0xFFF0_8000	0xFFF0_FFFF	32	Reserved
0xFFF1_0000	0xFFF1_3FFF	16	MPU
0xFFF1_4000	0xFFF1_BFFF	32	Reserved
0xFFF1_C000	0xFFF1_FFFF	16	CSE
0xFFF2_0000	0xFFF2_3FFF	16	Reserved

Table 7. SPC564Bxx and SPC56ECxx memory map (continued)

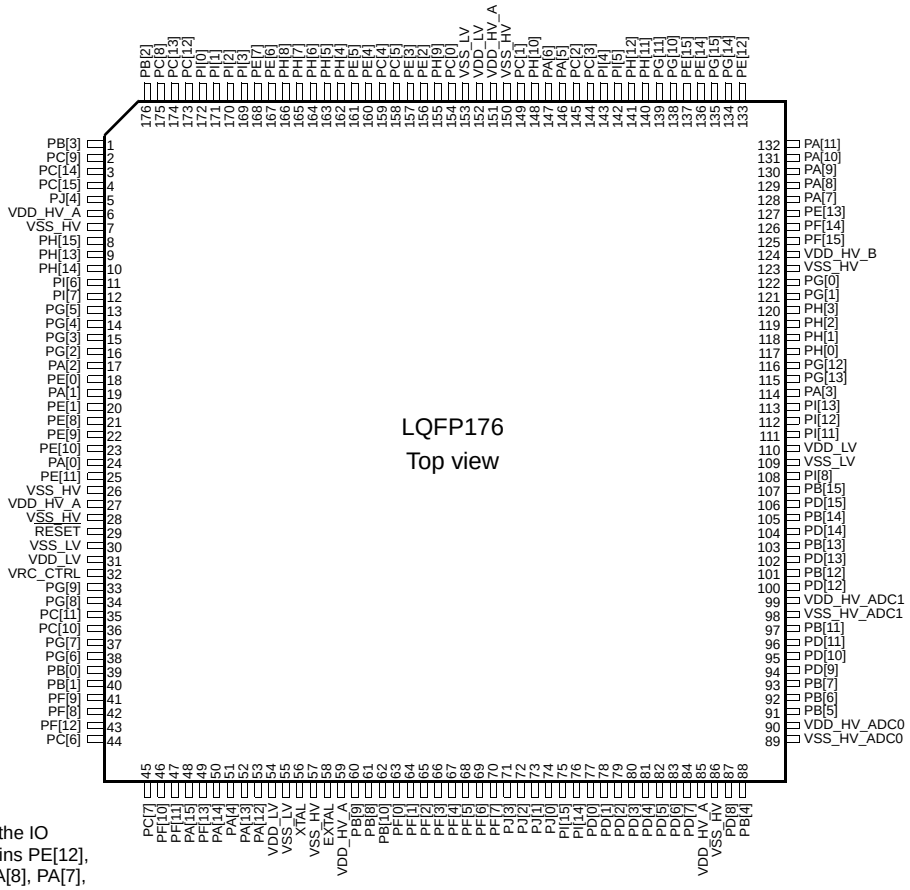
Start address	End address	Size (KB)	Region name
0xFFF2_4000	0xFFF2_7FFF	16	Semaphore
0xFFF2_8000	0xFFF3_7FFF	64	Reserved
0xFFF3_8000	0xFFF3_BFFF	16	SWT
0xFFF3_C000	0xFFF3_FFFF	16	STM
0xFFF4_0000	0xFFF4_3FFF	16	ECSM
0xFFF4_4000	0xFFF4_7FFF	16	eDMA
0xFFF4_8000	0xFFF4_BFFF	16	INTC
0xFFF4_C000	0xFFF4_FFFF	16	FEC
0xFFF5_0000	0xFFF8_FFFF	256	Reserved
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_0
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_1
0xFFF9_8000	0xFFF9_BFFF	16	DSPI_2
0xFFF9_C000	0xFFF9_FFFF	16	DSPI_3
0xFFFA_0000	0xFFFA_3FFF	16	DSPI_4
0xFFFA_4000	0xFFFA_7FFF	16	DSPI_5
0xFFFA_8000	0xFFFA_BFFF	16	DSPI_6
0xFFFA_C000	0xFFFA_FFFF	16	DSPI_7
0xFFFB_0000	0xFFFB_3FFF	16	LINFlexD_8
0xFFFB_4000	0xFFFB_7FFF	16	LINFlexD_9
0xFFFB_8000	0xFFFB_FFFF	32	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	0xFFFD_BFFF	16	Reserved
0xFFFD_C000	0xFFFD_FFFF	16	DMA_MUX
0xFFFE_0000	0xFFFE_3FFF	16	FlexRay
0xFFFE_4000	0xFFFF_BFFF	96	Reserved
0xFFFF_C000	0xFFFF_FFFF	16	BAM

4 Signal Description

4.1 Package pinouts

The following figures show the location of the signals on the available device packages.

For more information on pin multiplexing on this device, see *Table 9* through *Table 10*.

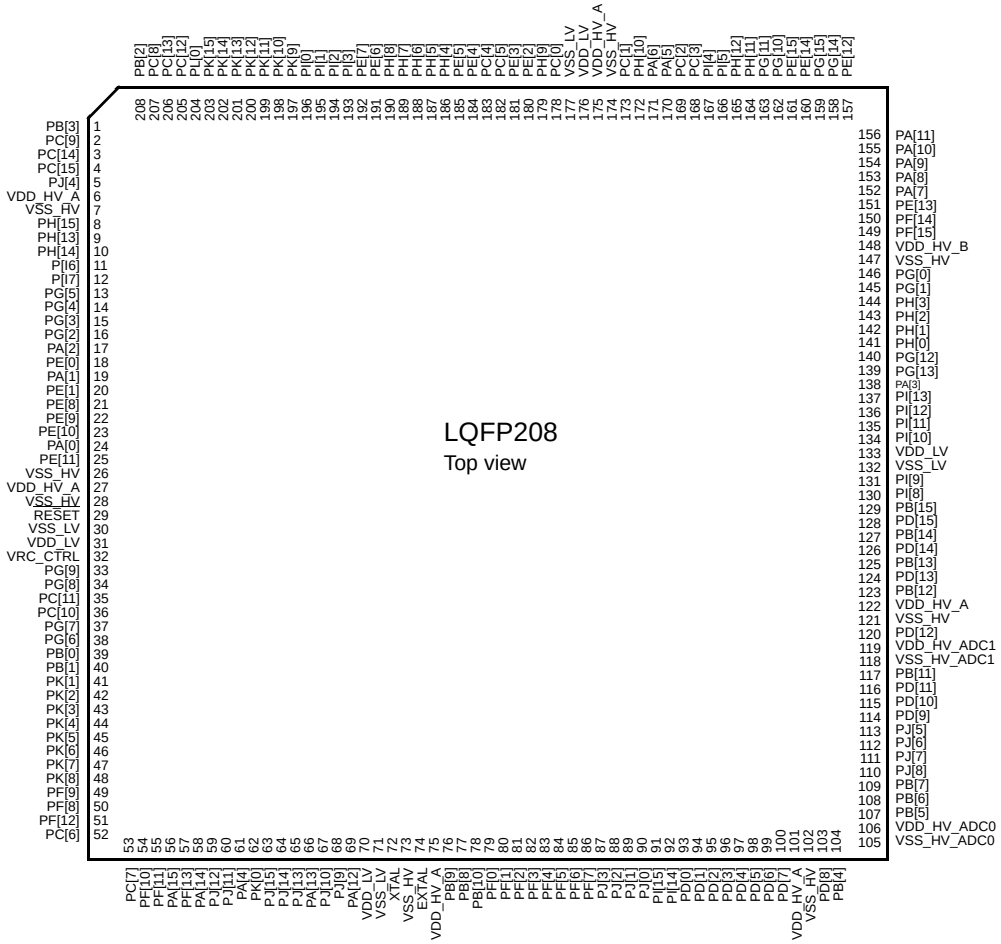


NOTE

1) VDD_HV_B supplies the IO voltage domain for the pins PE[12], PA[11], PA[10], PA[9], PA[8], PA[7], PE[13], PF[14], PF[15], PG[0], PG[1], PH[3], PH[2], PH[1], PH[0], PG[12], PG[13], and PA[3].

2)Availability of port pin alternate functions depends on product selection.

Figure 3. 176-pin LQFP configuration



NOTE
1) VDD_HV_B supplies the IO voltage domain for the pins PE[12], PA[11], PA[10], PA[9], PA[8], PA[7], PE[13], PF[14], PF[15], PG[0], PG[1], PH[3], PH[2], PH[1], PH[0], PG[12], PG[13], and PA[3].
2) Availability of port pin alternate functions depends on product selection.

Figure 4. 208-pin LQFP configuration

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
A	PC[15]	PB[2]	PC[13]	PI[1]	PE[7]	PH[8]	PE[2]	PE[4]	PC[4]	PE[3]	PH[9]	PI[4]	PH[11]	PE[14]	PA[10]	PG[11]	A
B	PH[13]	PC[14]	PC[8]	PC[12]	PI[3]	PE[6]	PH[5]	PE[5]	PC[5]	PC[0]	PC[2]	PH[12]	PG[10]	PA[11]	PA[9]	PA[8]	B
C	PH[14]	VDD_HV_A	PC[9]	PL[0]	PI[0]	PH[7]	PH[6]	VSS_LV	VDD_HV_A	PA[5]	PC[3]	PE[15]	PG[14]	PE[12]	PA[7]	PE[13]	C
D	PG[5]	PI[6]	PJ[4]	PB[3]	PK[15]	PI[2]	PH[4]	VDD_LV	PC[1]	PH[10]	PA[6]	PI[5]	PG[15]	PF[14]	PF[15]	PH[2]	D
E	PG[3]	PI[7]	PH[15]	PG[2]									PG[0]	PG[1]	PH[0]	VDD_HV_A	E
F	PA[2]	PG[4]	PA[1]	PE[1]									PH[1]	PH[3]	PG[12]	PG[13]	F
G	PE[8]	PE[0]	PE[10]	PA[0]			VSS_HV	VSS_HV	VSS_HV	VSS_HV			VDD_HV_B	PI[13]	PI[12]	PA[3]	G
H	PE[9]	VDD_HV_A	PE[11]	PK[1]			VSS_LV	VSS_HV	VSS_HV	VSS_HV			VDD_HV_A	VDD_LV	VSS_LV	PI[11]	H
J	VSS_HV	VRC_CTRL	VDD_LV	PG[9]			VSS_LV	VSS_LV	VSS_HV	VSS_HV			PD[15]	PI[8]	PI[9]	PI[10]	J
K	RESET	VSS_LV	PG[8]	PC[11]			VSS_LV	VSS_LV	VSS_LV	VDD_LV			PD[14]	PD[13]	PB[14]	PB[15]	K
L	PC[10]	PG[7]	PB[0]	PK[2]									PD[12]	PB[12]	PB[13]	VDD_HV_ADC1	L
M	PG[6]	PB[1]	PK[4]	PF[9]									PB[11]	PD[10]	PD[11]	VSS_HV_ADC1	M
N	PK[3]	PF[8]	PC[6]	PC[7]	PJ[13]	VDD_HV_A	PB[10]	PF[6]	VDD_HV_A	PJ[1]	PD[2]	PJ[5]	PB[5]	PB[6]	PJ[6]	PD[9]	N
P	PF[12]	PF[10]	PF[13]	PA[14]	PJ[9]	PA[12]	PF[0]	PF[5]	PF[7]	PJ[3]	PI[15]	PD[4]	PD[7]	PD[8]	PJ[8]	PJ[7]	P
R	PF[11]	PA[15]	PJ[11]	PJ[15]	PA[13]	PF[2]	PF[3]	PF[4]	VDD_LV	PJ[2]	PJ[0]	PD[0]	PD[3]	PD[6]	VDD_HV_ADC0	PB[7]	R
T	PJ[12]	PA[4]	PK[0]	PJ[14]	PJ[10]	PF[1]	XTAL	EXTAL	VSS_LV	PB[9]	PB[8]	PI[14]	PD[1]	PD[5]	VSS_HV_ADC0	PB[4]	T
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	

Notes:
1) VDD_HV_B supplies the IO voltage domain for the pins PE[12], PA[11], PA[10], PA[9], PA[8], PA[7], PE[13], PF[14], PF[15], PG[0], PG[1], PH[3], PH[2], PH[1], PH[0], PG[12], PG[13], and PA[3].
2) Availability of port pin alternate functions depends on product selection.

Figure 5. LBGA208 configuration

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
A	PC[15]	PB[2]	PC[13]	PI[1]	PE[7]	PH[8]	PE[2]	PE[4]	PC[4]	PE[3]	PH[9]	PI[4]	PH[11]	PE[14]	PA[10]	PG[11]	A
B	PH[13]	PC[14]	PC[8]	PC[12]	PI[3]	PE[6]	PH[5]	PE[5]	PC[5]	PC[0]	PC[2]	PH[12]	PG[10]	PA[11]	PA[9]	PA[8]	B
C	PH[14]	VDD_HV_A	PC[9]	PL[0]	PI[0]	PH[7]	PH[6]	VSS_LV	VDD_HV_A	PA[5]	PC[3]	PE[15]	PG[14]	PE[12]	PA[7]	PE[13]	C
D	PG[5]	PI[6]	PJ[4]	PB[3]	PK[15]	PI[2]	PH[4]	VDD_LV	PC[1]	PH[10]	PA[6]	PI[5]	PG[15]	PF[14]	PF[15]	PH[2]	D
E	PG[3]	PI[7]	PH[15]	PG[2]	VDD_LV	VSS_LV	PK[10]	PK[9]	PM[1]	PM[0]	PL[15]	PL[14]	PG[0]	PG[1]	PH[0]	VDD_HV_A	E
F	PA[2]	PG[4]	PA[1]	PE[1]	PL[2]	PM[6]	PL[1]	PK[11]	PM[5]	PL[13]	PL[12]	PM[2]	PH[1]	PH[3]	PG[12]	PG[13]	F
G	PE[8]	PE[0]	PE[10]	PA[0]	PL[3]	VSS_HV	VSS_HV	VSS_HV	VSS_HV	VSS_HV	VSS_HV	PK[12]	VDD_HV_B	PI[13]	PI[12]	PA[3]	G
H	PE[9]	VDD_HV_A	PE[11]	PK[1]	PL[4]	VSS_LV	VSS_LV	VSS_HV	VSS_HV	VSS_HV	VSS_HV	PK[13]	VDD_HV_A	VDD_LV	VSS_LV	PI[11]	H
J	VSS_HV	VRC_CTRL	VDD_LV	PG[9]	PL[5]	VSS_LV	VSS_LV	VSS_LV	VSS_HV	VSS_HV	VSS_HV	PK[14]	PD[15]	PI[8]	PI[9]	PI[10]	J
K	RESET	VSS_LV	PG[8]	PC[11]	PL[6]	VSS_LV	VSS_LV	VSS_LV	VSS_LV	VDD_LV	VDD_LV	PM[3]	PD[14]	PD[13]	PB[14]	PB[15]	K
L	PC[10]	PG[7]	PB[0]	PK[2]	PL[7]	VSS_LV	VSS_LV	VSS_LV	VSS_LV	VDD_LV	VDD_LV	PM[4]	PD[12]	PB[12]	PB[13]	VDD_HV_ADC1	L
M	PG[6]	PB[1]	PK[4]	PF[9]	PK[5]	PK[6]	PK[7]	PK[8]	PL[8]	PL[9]	PL[10]	PL[11]	PB[11]	PD[10]	PD[11]	VSS_HV_ADC1	M
N	PK[3]	PF[8]	PC[6]	PC[7]	PJ[13]	VDD_HV_A	PB[10]	PF[6]	VDD_HV_A	PJ[1]	PD[2]	PJ[5]	PB[5]	PB[6]	PJ[6]	PD[9]	N
P	PF[12]	PF[10]	PF[13]	PA[14]	PJ[9]	PA[12]	PF[0]	PF[5]	PF[7]	PJ[3]	PI[15]	PD[4]	PD[7]	PD[8]	PJ[8]	PJ[7]	P
R	PF[11]	PA[15]	PJ[11]	PJ[15]	PA[13]	PF[2]	PF[3]	PF[4]	VDD_LV	PJ[2]	PJ[0]	PD[0]	PD[3]	PD[6]	VDD_HV_ADC0	PB[7]	R
T	PJ[12]	PA[4]	PK[0]	PJ[14]	PJ[10]	PF[1]	XTAL	EXTAL	VSS_LV	PB[9]	PB[8]	PI[14]	PD[1]	PD[5]	VSS_HV_ADC0	PB[4]	T
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	

Notes: 1) VDD_HV_B supplies the IO voltage domain for the pins PE[12], PA[11], PA[10], PA[9], PA[8], PA[7], PE[13], PF[14], PF[15], PG[0], PG[1], PH[3], PH[2], PH[1], PH[0], PG[12], PG[13], PA[3], PM[3], and PM[4].
2)Availability of port pin alternate functions depends on product selection.

Figure 6. 256-pin BGA configuration

4.2 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 tristate with the following exceptions:

- PA[9] (FAB) is pull-down. Without external strong pull-up the device starts fetching from flash.
- PA[8], PC[0], PL[8], and PH[9:10] are in input weak pull-up when out of reset.
- RESET pad is driven low. This is released only after PHASE2 reset completion.

4.3 Pad configuration during standby mode exit

During standby mode exit, all pads are configured as reset mode exit with the exception of low power wakeup pads (PA[0, 1, 2, 4, 15], PB[1, 3, 8, 9, 10], PC[7, 9, 11], PD[0, 1], PE[0, 3, 5, 9, 11], PF[9, 11, 13],

PG[3, 5, 7, 21], PI[1, 3], and PJ[13](b), which will be configured according to their respective configuration done in wakeup module.

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 8. Voltage supply pin descriptions

Port pin	Function	Pin number			
		LQFP176	LQFP208	LBGA208	LBGA256
VDD_HV_A	Digital supply voltage for I/O domain A	6, 27, 59, 85, 151	6, 27, 75, 101, 122, 175	C2, C9, H2, N6, N9, H13, E16	C2, C9, H2, N6, N9, H13, E16
VDD_HV_B	Digital supply voltage for I/O domain B	124	148	G13	G13
VSS_HV	Digital ground	7, 26, 28, 57, 86, 123, 150	7, 26, 28, 73, 102, 121, 147, 174	G7, G8, G9, G10, H8, H9, H10, J1, J9, J10	G6, G7, G8, G9, G10, G11, H8, H9, H10, H11, J1, J9, J10, J11
VDD_LV(1)	1.2 V supply pins	31, 54, 110, 152	31, 70, 133, 176	D8, K10, H14, J3, R9	J3, E5, D8, H14, K10, K11, L10, L11, R9
VSS_LV(1)	1.2 V supply pins	30, 55, 109, 153	30, 71, 132, 177	C8, H7, H15, J7, J8, K7, K2, K8, K9, T9	K2, C8, E6, H6, H7, J6, K6, L6, J7, K7, L7, J8, K8, L8, K9, L9, H15, T9

b. PJ[13] is not available on 176 pin package. However, it should be configured with weak pull device to minimize leakage. See the WKPU chapter for details.

Table 8. Voltage supply pin descriptions (continued)

Port pin	Function	Pin number			
		LQFP176	LQFP208	LBGA208	LBGA256
VRC_CTRL(2)	Base control voltage for external NPN device	32	32	J2	J2
VSS_HV_ADC0	Reference ground and analog ground for the A/D converter 0 (10 bit)	89	105	T15	T15
VDD_HV_ADC0	Reference voltage and analog supply for the A/D converter 0 (10 bit)	90	106	R15	R15
VSS_HV_ADC1	Reference ground and analog ground for the A/D converter 1 (12 bit)	98	118	M16	M16
VDD_HV_ADC1	Reference voltage and analog supply for the A/D converter 1 (12-bit)	99	119	L16	L16

1. Decoupling capacitor must be connected between each VDD_LV/VSS_LV supply pair to ensure stable voltage.
2. This voltage is generated by the device and no external voltage should be supplied.

4.5 Pad types

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

S = Slow(c)

M = Medium(c), (d)

F = Fast(c), (d)

I = Input only with analog feature(c)

A = Analog

4.6 System pins

The system pins are listed in *Table 9*.

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

d. All medium and fast pads are in slow configuration by default at reset and can be configured as fast or medium. For example, Fast/Medium pad will be Medium by default at reset. Similarly, Slow/Medium pad will be Slow by default. Only exception is PC[1] which is in medium configuration by default (refer to PCR.SRC in the reference manual, Pad Configuration Registers (PCR0—PCR198)).

Table 9. System pin descriptions

Port pin	Function	I/O direction	Pad type	RESET config.	Pin number			
					LQFP 176	LQFP 208	LBGA208	LBGA 256
RESET	Bidirectional reset with Schmitt-Trigger characteristics and noise filter.	I/O	M	Input, weak pull-up only after PHASE2	29	29	K1	K1
EXTAL	Analog input of the oscillator amplifier circuit. Needs to be grounded if oscillator bypass mode is used.	I	A(1)	—	58	74	T8	T8
XTAL	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.	I/O	A(1)	—	56	72	T7	T7

1. For analog pads, it is not recommended to enable IBE if APC is enabled to avoid extra current in middle range voltage.

4.7 Functional ports

The functional port pins are listed in *Table 10*.

Table 10. Functional port pin descriptions

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PA[0]	PCR[0]	AF0 AF1 AF2 AF3 — —	GPIO[0] E0UC[0] CLKOUT E0UC[13] WKPU[19] CAN1RX	SIUL eMIOS_0 MC_CGM eMIOS_0 WKPU FlexCAN_1	I/O I/O O I/O I I	M/S	Tristate	24	24	G4	G4
PA[1]	PCR[1]	AF0 AF1 AF2 AF3 — — —	GPIO[1] E0UC[1] — — WKPU[2] CAN3RX NMI[0][3]	SIUL eMIOS_0 — — WKPU FlexCAN_3 WKPU	I/O I/O — — I I I	S	Tristate	19	19	F3	F3
PA[2]	PCR[2]	AF0 AF1 AF2 AF3 — —	GPIO[2] E0UC[2] — MA[2] WKPU[3] NMI[1][3]	SIUL eMIOS_0 — ADC_0 WKPU WKPU	I/O I/O — O I I	S	Tristate	17	17	F1	F1
PA[3]	PCR[3]	AF0 AF1 AF2 AF3 — — —	GPIO[3] E0UC[3] LIN5TX CS4_1 RX_ER_CLK EIRQ[0] ADC1_S[0]	SIUL eMIOS_0 LINFlexD_5 DSPI_1 FEC SIUL ADC_1	I/O I/O O O I I I	M/S	Tristate	114	138	G16	G16
PA[4]	PCR[4]	AF0 AF1 AF2 AF3 — —	GPIO[4] E0UC[4] — CS0_1 LIN5RX WKPU[9]	SIUL eMIOS_0 — DSPI_1 LINFlexD_5 WKPU	I/O I/O — I/O I I	S	Tristate	51	61	T2	T2
PA[5]	PCR[5]	AF0 AF1 AF2	GPIO[5] E0UC[5] LIN4TX	SIUL eMIOS_0 LINFlexD_4	I/O I/O O	M/S	Tristate	146	170	C10	C10

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PA[6]	PCR[6]	AF0 AF1 AF2 AF3 — —	GPIO[6] E0UC[6] — CS1_1 LIN4RX EIRQ[1]	SIUL eMIOS_0 — DSPI_1 LINFlexD_4 SIUL	I/O I/O — O I I	S	Tristate	147	171	D11	D11
PA[7]	PCR[7]	AF0 AF1 AF2 AF3 — — —	GPIO[7] E0UC[7] LIN3TX — RXD[2] EIRQ[2] ADC1_S[1]	SIUL eMIOS_0 LINFlexD_3 — FEC SIUL ADC_1	I/O I/O O — I I I	M/S	Tristate	128	152	C15	C15
PA[8]	PCR[8]	AF0 AF1 AF2 AF3 — — — —	GPIO[8] E0UC[8] E0UC[14] — RXD[1] EIRQ[3] ABS[0] LIN3RX	SIUL eMIOS_0 eMIOS_0 — FEC SIUL MC_RGM LINFlexD_3	I/O I/O I/O — I I I I	M/S	Input, weak pull-up	129	153	B16	B16
PA[9]	PCR[9]	AF0 AF1 AF2 AF3 — —	GPIO[9] E0UC[9] — CS2_1 RXD[0] FAB	SIUL eMIOS_0 — DSPI1 FEC MC_RGM	I/O I/O — O I I	M/S	Pull-down	130	154	B15	B15
PA[10]	PCR[10]	AF0 AF1 AF2 AF3 — — —	GPIO[10] E0UC[10] SDA LIN2TX COL ADC1_S[2] SIN_1	SIUL eMIOS_0 I^2C LINFlexD_2 FEC ADC_1 DSPI_1	I/O I/O I/O O I I I	M/S	Tristate	131	155	A15	A15

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PA[11]	PCR[11]	AF0 AF1 AF2 AF3 — — — —	GPIO[11] E0UC[11] SCL — RX_ER EIRQ[16] LIN2RX ADC1_S[3]	SIUL eMIOS_0 I^2C — FEC SIUL LINFlexD_2 ADC_1	I/O I/O I/O — I I I I	M/S	Tristate	132	156	B14	B14
PA[12]	PCR[12]	AF0 AF1 AF2 AF3 — —	GPIO[12] — E0UC[28] CS3_1 EIRQ[17] SIN_0	SIUL — eMIOS_0 DSPI1 SIUL DSPI_0	I/O — I/O O I I	S	Tristate	53	69	P6	P6
PA[13]	PCR[13]	AF0 AF1 AF2 AF3	GPIO[13] SOUT_0 E0UC[29] —	SIUL DSPI_0 eMIOS_0 —	I/O O I/O —	M/S	Tristate	52	66	R5	R5
PA[14]	PCR[14]	AF0 AF1 AF2 AF3 —	GPIO[14] SCK_0 CS0_0 E0UC[0] EIRQ[4]	SIUL DSPI_0 DSPI_0 eMIOS_0 SIUL	I/O I/O I/O I/O I	M/S	Tristate	50	58	P4	P4
PA[15]	PCR[15]	AF0 AF1 AF2 AF3 —	GPIO[15] CS0_0 SCK_0 E0UC[1] WKPU[10]	SIUL DSPI_0 DSPI_0 eMIOS_0 WKPU	I/O I/O I/O I/O I	M/S	Tristate	48	56	R2	R2
PB[0]	PCR[16]	AF0 AF1 AF2 AF3	GPIO[16] CAN0TX E0UC[30] LIN0TX	SIUL FlexCAN_0 eMIOS_0 LINFlexD_0	I/O O I/O I	M/S	Tristate	39	39	L3	L3

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PB[1]	PCR[17]	AF0 AF1 AF2 — — —	GPIO[17] — E0UC[31] LIN0RX WKPU[4] CAN0RX	SIUL — eMIOS_0 LINFlexD_0 WKPU FlexCAN_0	I/O — I/O I I I	S	Tristate	40	40	M2	M2
PB[2]	PCR[18]	AF0 AF1 AF2 AF3	GPIO[18] LIN0TX SDA E0UC[30]	SIUL LINFlexD_0 I^2C eMIOS_0	I/O O I/O I/O	M/S	Tristate	176	208	A2	A2
PB[3]	PCR[19]	AF0 AF1 AF2 AF3 — —	GPIO[19] E0UC[31] SCL — WKPU[11] LIN0RX	SIUL eMIOS_0 I^2C — WKPU LINFlexD_0	I/O I/O I/O — I I	S	Tristate	1	1	D4	D4
PB[4]	PCR[20]	AF0 AF1 AF2 AF3 — —	GPI[20] — — — ADC0_P[0] ADC1_P[0]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	88	104	T16	T16
PB[5]	PCR[21]	AF0 AF1 AF2 AF3 — —	GPI[21] — — — ADC0_P[1] ADC1_P[1]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	91	107	N13	N13
PB[6]	PCR[22]	AF0 AF1 AF2 AF3 — —	GPI[22] — — — ADC0_P[2] ADC1_P[2]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	92	108	N14	N14

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PB[7]	PCR[23]	AF0 AF1 AF2 AF3 — —	GPI[23] — — — ADC0_P[3] ADC1_P[3]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	93	109	R16	R16
PB[8]	PCR[24]	AF0 AF1 AF2 AF3 — — — —	GPI[24] — — — ADC0_S[0] ADC1_S[4] WKPU[25] OSC32k_XTAL[4]	SIUL — — — ADC_0 ADC_1 WKPU SXOSC	I — — — I I I I	I	—	61	77	T11	T11
PB[9][5]	PCR[25]	AF0 AF1 AF2 AF3 — — — —	GPI[25] — — — ADC0_S[1] ADC1_S[5] WKPU[26] OSC32k_EXTAL[4]	SIUL — — — ADC_0 ADC_1 WKPU SXOSC	I — — — I I I I	I	—	60	76	T10	T10
PB[10]	PCR[26]	AF0 AF1 AF2 AF3 — — —	GPIO[26] SOUT_1 CAN3TX — ADC0_S[2] ADC1_S[6] WKPU[8]	SIUL DSPI_1 FlexCAN_3 — ADC_0 ADC_1 WKPU	I/O O — — I I I	S	Tristate	62	78	N7	N7
PB[11]	PCR[27]	AF0 AF1 AF2 AF3 —	GPIO[27] E0UC[3] — CS0_0 ADC0_S[3]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — I/O I	S	Tristate	97	117	M13	M13

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PB[12]	PCR[28]	AF0 AF1 AF2 AF3 —	GPIO[28] E0UC[4] — CS1_0 ADC0_X[0]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — O I	S	Tristate	101	123	L14	L14
PB[13]	PCR[29]	AF0 AF1 AF2 AF3 —	GPIO[29] E0UC[5] — CS2_0 ADC0_X[1]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — O I	S	Tristate	103	125	L15	L15
PB[14]	PCR[30]	AF0 AF1 AF2 AF3 —	GPIO[30] E0UC[6] — CS3_0 ADC0_X[2]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — O I	S	Tristate	105	127	K15	K15
PB[15]	PCR[31]	AF0 AF1 AF2 AF3 —	GPIO[31] E0UC[7] — CS4_0 ADC0_X[3]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — O I	S	Tristate	107	129	K16	K16
PC[0] (6)	PCR[32]	AF0 AF1 AF2 AF3	GPIO[32] — TDI —	SIUL — JTAGC —	I/O — I —	M/S	Input, weak pull-up	154	178	B10	B10
PC[1] (6)	PCR[33]	AF0 AF1 AF2 AF3	GPIO[33] — TDO —	SIUL — JTAGC —	I/O — O —	F/M	Tristate	149	173	D9	D9
PC[2]	PCR[34]	AF0 AF1 AF2 AF3 —	GPIO[34] SCK_1 CAN4TX — EIRQ[5]	SIUL DSPI_1 FlexCAN_4 — SIUL	I/O I/O O — I	M/S	Tristate	145	169	B11	B11

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PC[3]	PCR[35]	AF0 AF1 AF2 AF3 — — —	GPIO[35] CS0_1 MA[0] — CAN1RX CAN4RX EIRQ[6]	SIUL DSPI_1 ADC_0 — FlexCAN_1 FlexCAN_4 SIUL	I/O I/O O I I I	S	Tristate	144	168	C11	C11
PC[4]	PCR[36]	AF0 AF1 AF2 AF3 ALT4 — — —	GPIO[36] E1UC[31] — FR_B_TX_EN SIN_1 CAN3RX EIRQ[18]	SIUL eMIOS_1 — Flexray DSPI_1 FlexCAN_3 SIUL	I/O I/O — O I I I	M/S	Tristate	159	183	A9	A9
PC[5]	PCR[37]	AF0 AF1 AF2 AF3 ALT4 —	GPIO[37] SOUT_1 CAN3TX — FR_A_TX EIRQ[7]	SIUL DSPI_1 FlexCAN_3 — Flexray SIUL	I/O O O — O I	M/S	Tristate	158	182	B9	B9
PC[6]	PCR[38]	AF0 AF1 AF2 AF3	GPIO[38] LIN1TX E1UC[28] —	SIUL LINFlexD_1 eMIOS_1 —	I/O O I/O —	S	Tristate	44	52	N3	N3
PC[7]	PCR[39]	AF0 AF1 AF2 AF3 — —	GPIO[39] — E1UC[29] — LIN1RX WKPU[12]	SIUL — eMIOS_1 — LINFlexD_1 WKPU	I/O — I/O — I I	S	Tristate	45	53	N4	N4
PC[8]	PCR[40]	AF0 AF1 AF2 AF3	GPIO[40] LIN2TX E0UC[3] —	SIUL LINFlexD_2 eMIOS_0 —	I/O O I/O —	S	Tristate	175	207	B3	B3

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PC[9]	PCR[41]	AF0 AF1 AF2 AF3 — —	GPIO[41] — E0UC[7] — LIN2RX WKPU[13]	SIUL — eMIOS_0 — LINFlexD_2 WKPU	I/O — I/O — I I	S	Tristate	2	2	C3	C3
PC[10]	PCR[42]	AF0 AF1 AF2 AF3	GPIO[42] CAN1TX CAN4TX MA[1]	SIUL FlexCAN_1 FlexCAN_4 ADC_0	I/O O O O	M/S	Tristate	36	36	L1	L1
PC[11]	PCR[43]	AF0 AF1 AF2 AF3 — — —	GPIO[43] — — MA[2] CAN1RX CAN4RX WKPU[5]	SIUL — — ADC_0 FlexCAN_1 FlexCAN_4 WKPU	I/O — — O I I I	S	Tristate	35	35	K4	K4
PC[12]	PCR[44]	AF0 AF1 AF2 AF3 ALT4 — —	GPIO[44] E0UC[12] — — FR_DBG[0] SIN_2 EIRQ[19]	SIUL eMIOS_0 — — Flexray DSPI_2 SIUL	I/O I/O — — O I I	M/S	Tristate	173	205	B4	B4
PC[13]	PCR[45]	AF0 AF1 AF2 AF3 ALT4	GPIO[45] E0UC[13] SOUT_2 — FR_DBG[1]	SIUL eMIOS_0 DSPI_2 — Flexray	I/O I/O O — O	M/S	Tristate	174	206	A3	A3
PC[14]	PCR[46]	AF0 AF1 AF2 AF3 ALT4 —	GPIO[46] E0UC[14] SCK_2 — FR_DBG[2] EIRQ[8]	SIUL eMIOS_0 DSPI_2 — Flexray SIUL	I/O I/O I/O — O I	M/S	Tristate	3	3	B2	B2

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PC[15]	PCR[47]	AF0 AF1 AF2 AF3 ALT4	GPIO[47] E0UC[15] CS0_2 — FR_DBG[3] EIRQ[20]	SIUL eMIOS_0 DSPI_2 — Flexray SIUL	I/O I/O I/O — O I	M/S	Tristate	4	4	A1	A1
PD[0]	PCR[48]	AF0 AF1 AF2 AF3 — — —	GPI[48] — — — ADC0_P[4] ADC1_P[4] WKPU[27]	SIUL — — — ADC_0 ADC_1 WKPU	I — — — I I I	I	Tristate	77	93	R12	R12
PD[1]	PCR[49]	AF0 AF1 AF2 AF3 — — —	GPI[49] — — — ADC0_P[5] ADC1_P[5] WKPU[28]	SIUL — — — ADC_0 ADC_1 WKPU	I — — — I I I	I	Tristate	78	94	T13	T13
PD[2]	PCR[50]	AF0 AF1 AF2 AF3 — —	GPI[50] — — — ADC0_P[6] ADC1_P[6]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	79	95	N11	N11
PD[3]	PCR[51]	AF0 AF1 AF2 AF3 — —	GPI[51] — — — ADC0_P[7] ADC1_P[7]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	80	96	R13	R13

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PD[4]	PCR[52]	AF0 AF1 AF2 AF3 — —	GPI[52] — — — ADC0_P[8] ADC1_P[8]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	81	97	P12	P12
PD[5]	PCR[53]	AF0 AF1 AF2 AF3 — —	GPI[53] — — — ADC0_P[9] ADC1_P[9]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	82	98	T14	T14
PD[6]	PCR[54]	AF0 AF1 AF2 AF3 — —	GPI[54] — — — ADC0_P[10] ADC1_P[10]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	83	99	R14	R14
PD[7]	PCR[55]	AF0 AF1 AF2 AF3 — —	GPI[55] — — — ADC0_P[11] ADC1_P[11]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	84	100	P13	P13
PD[8]	PCR[56]	AF0 AF1 AF2 AF3 — —	GPI[56] — — — ADC0_P[12] ADC1_P[12]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	87	103	P14	P14
PD[9]	PCR[57]	AF0 AF1 AF2 AF3 — —	GPI[57] — — — ADC0_P[13] ADC1_P[13]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	94	114	N16	N16

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PD[10]	PCR[58]	AF0 AF1 AF2 AF3 — —	GPI[58] — — — ADC0_P[14] ADC1_P[14]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	95	115	M14	M14
PD[11]	PCR[59]	AF0 AF1 AF2 AF3 — —	GPI[59] — — — ADC0_P[15] ADC1_P[15]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	96	116	M15	M15
PD[12]	PCR[60]	AF0 AF1 AF2 AF3 —	GPIO[60] CS5_0 E0UC[24] — ADC0_S[4]	SIUL DSPI_0 eMIOS_0 — ADC_0	I/O O I/O — I	S	Tristate	100	120	L13	L13
PD[13]	PCR[61]	AF0 AF1 AF2 AF3 —	GPIO[61] CS0_1 E0UC[25] — ADC0_S[5]	SIUL DSPI_1 eMIOS_0 — ADC_0	I/O I/O I/O — I	S	Tristate	102	124	K14	K14
PD[14]	PCR[62]	AF0 AF1 AF2 AF3 ALT4 —	GPIO[62] CS1_1 E0UC[26] — FR_DBG[0] ADC0_S[6]	SIUL DSPI_1 eMIOS_0 — Flexray ADC_0	I/O O I/O — O I	S	Tristate	104	126	K13	K13
PD[15]	PCR[63]	AF0 AF1 AF2 AF3 ALT4 —	GPIO[63] CS2_1 E0UC[27] — FR_DBG[1] ADC0_S[7]	SIUL DSPI_1 eMIOS_0 — Flexray ADC_0	I/O O I/O — O I	S	Tristate	106	128	J13	J13

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PE[0]	PCR[64]	AF0 AF1 AF2 AF3 — —	GPIO[64] E0UC[16] — — CAN5RX WKPU[6]	SIUL eMIOS_0 — — FlexCAN_5 WKPU	I/O I/O — — I I	S	Tristate	18	18	G2	G2
PE[1]	PCR[65]	AF0 AF1 AF2 AF3	GPIO[65] E0UC[17] CAN5TX —	SIUL eMIOS_0 FlexCAN_5 —	I/O I/O O —	M/S	Tristate	20	20	F4	F4
PE[2]	PCR[66]	AF0 AF1 AF2 AF3 ALT4 — —	GPIO[66] E0UC[18] — — FR_A_TX_EN SIN_1 EIRQ[21]	SIUL eMIOS_0 — — Flexray DSPI_1 SIUL	I/O I/O — — O I I	M/S	Tristate	156	180	A7	A7
PE[3]	PCR[67]	AF0 AF1 AF2 AF3 — —	GPIO[67] E0UC[19] SOUT_1 — FR_A_RX WKPU[29]	SIUL eMIOS_0 DSPI_1 — Flexray WKPU	I/O I/O O — I I	M/S	Tristate	157	181	A10	A10
PE[4]	PCR[68]	AF0 AF1 AF2 AF3 ALT4 —	GPIO[68] E0UC[20] SCK_1 — FR_B_TX EIRQ[9]	SIUL eMIOS_0 DSPI_1 — Flexray SIUL	I/O I/O I/O — O I	M/S	Tristate	160	184	A8	A8
PE[5]	PCR[69]	AF0 AF1 AF2 AF3 — —	GPIO[69] E0UC[21] CS0_1 MA[2] FR_B_RX WKPU[30]	SIUL eMIOS_0 DSPI_1 ADC_0 Flexray WKPU	I/O I/O I/O O I I	M/S	Tristate	161	185	B8	B8

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PE[6]	PCR[70]	AF0 AF1 AF2 AF3 —	GPIO[70] E0UC[22] CS3_0 MA[1] EIRQ[22]	SIUL eMIOS_0 DSPI_0 ADC_0 SIUL	I/O I/O O O I	M/S	Tristate	167	191	B6	B6
PE[7]	PCR[71]	AF0 AF1 AF2 AF3 —	GPIO[71] E0UC[23] CS2_0 MA[0] EIRQ[23]	SIUL eMIOS_0 DSPI_0 ADC_0 SIUL	I/O I/O O O I	M/S	Tristate	168	192	A5	A5
PE[8]	PCR[72]	AF0 AF1 AF2 AF3	GPIO[72] CAN2TX E0UC[22] CAN3TX	SIUL FlexCAN_2 eMIOS_0 FlexCAN_3	I/O O I/O O	M/S	Tristate	21	21	G1	G1
PE[9]	PCR[73]	AF0 AF1 AF2 AF3 — — —	GPIO[73] — E0UC[23] — WKPU[7] CAN2RX CAN3RX	SIUL — eMIOS_0 — WKPU FlexCAN_2 FlexCAN_3	I/O — I/O — I I I	S	Tristate	22	22	H1	H1
PE[10]	PCR[74]	AF0 AF1 AF2 AF3 —	GPIO[74] LIN3TX CS3_1 E1UC[30] EIRQ[10]	SIUL LINFlexD_3 DSPI_1 eMIOS_1 SIUL	I/O O O I/O I	S	Tristate	23	23	G3	G3
PE[11]	PCR[75]	AF0 AF1 AF2 AF3 — —	GPIO[75] E0UC[24] CS4_1 — LIN3RX WKPU[14]	SIUL eMIOS_0 DSPI_1 — LINFlexD_3 WKPU	I/O I/O O — I I	S	Tristate	25	25	H3	H3

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PE[12]	PCR[76]	AF0 AF1 AF2 AF3 — — — —	GPIO[76] — E1UC[19] — CRS SIN_2 EIRQ[11] ADC1_S[7]	SIUL — eMIOS_1 — FEC DSPI_2 SIUL ADC_1	I/O — I/O — I I I I	M/S	Tristate	133	157	C14	C14
PE[13]	PCR[77]	AF0 AF1 AF2 AF3 —	GPIO[77] SOUT_2 E1UC[20] — RXD[3]	SIUL DSPI_2 eMIOS_1 — FEC	I/O O I/O — I	M/S	Tristate	127	151	C16	C16
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	M/S	Tristate	136	160	A14	A14
PE[15]	PCR[79]	AF0 AF1 AF2 AF3	GPIO[79] CS0_2 E1UC[22] SCK_6	SIUL DSPI_2 eMIOS_1 DSPI_6	I/O I/O I/O I/O	M/S	Tristate	137	161	C12	C12
PF[0]	PCR[80]	AF0 AF1 AF2 AF3 —	GPIO[80] E0UC[10] CS3_1 — ADC0_S[8]	SIUL eMIOS_0 DSPI_1 — ADC_0	I/O I/O O — I	S	Tristate	63	79	P7	P7
PF[1]	PCR[81]	AF0 AF1 AF2 AF3 —	GPIO[81] E0UC[11] CS4_1 — ADC0_S[9]	SIUL eMIOS_0 DSPI_1 — ADC_0	I/O I/O O — I	S	Tristate	64	80	T6	T6

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PF[2]	PCR[82]	AF0 AF1 AF2 AF3 —	GPIO[82] E0UC[12] CS0_2 — ADC0_S[10]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O I/O — I	S	Tristate	65	81	R6	R6
PF[3]	PCR[83]	AF0 AF1 AF2 AF3 —	GPIO[83] E0UC[13] CS1_2 — ADC0_S[11]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O O — I	S	Tristate	66	82	R7	R7
PF[4]	PCR[84]	AF0 AF1 AF2 AF3 —	GPIO[84] E0UC[14] CS2_2 — ADC0_S[12]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O O — I	S	Tristate	67	83	R8	R8
PF[5]	PCR[85]	AF0 AF1 AF2 AF3 —	GPIO[85] E0UC[22] CS3_2 — ADC0_S[13]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O O — I	S	Tristate	68	84	P8	P8
PF[6]	PCR[86]	AF0 AF1 AF2 AF3 —	GPIO[86] E0UC[23] CS1_1 — ADC0_S[14]	SIUL eMIOS_0 DSPI_1 — ADC_0	I/O I/O O — I	S	Tristate	69	85	N8	N8
PF[7]	PCR[87]	AF0 AF1 AF2 AF3 —	GPIO[87] — CS2_1 — ADC0_S[15]	SIUL — DSPI_1 — ADC_0	I/O — O — I	S	Tristate	70	86	P9	P9
PF[8]	PCR[88]	AF0 AF1 AF2 AF3	GPIO[88] CAN3TX CS4_0 CAN2TX	SIUL FlexCAN_3 DSPI_0 FlexCAN_2	I/O O O O	M/S	Tristate	42	50	N2	N2

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PF[9]	PCR[89]	AF0 AF1 AF2 AF3 — — —	GPIO[89] E1UC[1] CS5_0 — CAN2RX CAN3RX WKPU[22]	SIUL eMIOS_1 DSPI_0 — FlexCAN_2 FlexCAN_3 WKPU	I/O I/O O — I I I	S	Tristate	41	49	M4	M4
PF[10]	PCR[90]	AF0 AF1 AF2 AF3	GPIO[90] CS1_0 LIN4TX E1UC[2]	SIUL DSPI_0 LINFlexD_4 eMIOS_1	I/O O O I/O	M/S	Tristate	46	54	P2	P2
PF[11]	PCR[91]	AF0 AF1 AF2 AF3 — —	GPIO[91] CS2_0 E1UC[3] — LIN4RX WKPU[15]	SIUL DSPI_0 eMIOS_1 — LINFlexD_4 WKPU	I/O O I/O — I I	S	Tristate	47	55	R1	R1
PF[12]	PCR[92]	AF0 AF1 AF2 AF3	GPIO[92] E1UC[25] LIN5TX —	SIUL eMIOS_1 LINFlexD_5 —	I/O I/O O —	M/S	Tristate	43	51	P1	P1
PF[13]	PCR[93]	AF0 AF1 AF2 AF3 — —	GPIO[93] E1UC[26] — — LIN5RX WKPU[16]	SIUL eMIOS_1 — — LINFlexD_5 WKPU	I/O I/O — — I I	S	Tristate	49	57	P3	P3
PF[14]	PCR[94]	AF0 AF1 AF2 AF3 ALT4	GPIO[94] CAN4TX E1UC[27] CAN1TX MDIO	SIUL FlexCAN_4 eMIOS_1 FlexCAN_1 FEC	I/O O I/O O I/O	M/S	Tristate	126	150	D14	D14

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PF[15]	PCR[95]	AF0 AF1 AF2 AF3 — — — —	GPIO[95] E1UC[4] — — RX_DV CAN1RX CAN4RX EIRQ[13]	SIUL eMIOS_1 — — FEC FlexCAN_1 FlexCAN_4 SIUL	I/O I/O — — I I I I	M/S	Tristate	125	149	D15	D15
PG[0]	PCR[96]	AF0 AF1 AF2 AF3 ALT4	GPIO[96] CAN5TX E1UC[23] — MDC	SIUL FlexCAN_5 eMIOS_1 — FEC	I/O O I/O — O	F	Tristate	122	146	E13	E13
PG[1]	PCR[97]	AF0 AF1 AF2 AF3 — — —	GPIO[97] — E1UC[24] — TX_CLK CAN5RX EIRQ[14]	SIUL — eMIOS_1 — FEC FlexCAN_5 SIUL	I/O — I/O — I I I	M	Tristate	121	145	E14	E14
PG[2]	PCR[98]	AF0 AF1 AF2 AF3	GPIO[98] E1UC[11] SOUT_3 —	SIUL eMIOS_1 DSPI_3 —	I/O I/O O —	M/S	Tristate	16	16	E4	E4
PG[3]	PCR[99]	AF0 AF1 AF2 AF3 —	GPIO[99] E1UC[12] CS0_3 — WKPU[17]	SIUL eMIOS_1 DSPI_3 — WKPU	I/O I/O I/O — I	S	Tristate	15	15	E1	E1
PG[4]	PCR[100]	AF0 AF1 AF2 AF3	GPIO[100] E1UC[13] SCK_3 —	SIUL eMIOS_1 DSPI_3 —	I/O I/O I/O —	M/S	Tristate	14	14	F2	F2

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PG[5]	PCR[101]	AF0 AF1 AF2 AF3 — —	GPIO[101] E1UC[14] — — WKPU[18] SIN_3	SIUL eMIOS_1 — — WKPU DSPI_3	I/O I/O — — I I	S	Tristate	13	13	D1	D1
PG[6]	PCR[102]	AF0 AF1 AF2 AF3	GPIO[102] E1UC[15] LIN6TX —	SIUL eMIOS_1 LINFlexD_6 —	I/O I/O O —	M/S	Tristate	38	38	M1	M1
PG[7]	PCR[103]	AF0 AF1 AF2 AF3 — —	GPIO[103] E1UC[16] E1UC[30] — LIN6RX WKPU[20]	SIUL eMIOS_1 eMIOS_1 — LINFlexD_6 WKPU	I/O I/O I/O — I I	S	Tristate	37	37	L2	L2
PG[8]	PCR[104]	AF0 AF1 AF2 AF3 —	GPIO[104] E1UC[17] LIN7TX CS0_2 EIRQ[15]	SIUL eMIOS_1 LINFlexD_7 DSPI_2 SIUL	I/O I/O O I/O I	S	Tristate	34	34	K3	K3
PG[9]	PCR[105]	AF0 AF1 AF2 AF3 — —	GPIO[105] E1UC[18] — SCK_2 LIN7RX WKPU[21]	SIUL eMIOS_1 — DSPI_2 LINFlexD_7 WKPU	I/O I/O — I/O I I	S	Tristate	33	33	J4	J4
PG[10]	PCR[106]	AF0 AF1 AF2 AF3 —	GPIO[106] E0UC[24] E1UC[31] — SIN_4	SIUL eMIOS_0 eMIOS_1 — DSPI_4	I/O I/O I/O — I	S	Tristate	138	162	B13	B13
PG[11]	PCR[107]	AF0 AF1 AF2 AF3	GPIO[107] E0UC[25] CS0_4 CS0_6	SIUL eMIOS_0 DSPI_4 DSPI_6	I/O I/O I/O I/O	M/S	Tristate	139	163	A16	A16

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PG[12]	PCR[108]	AF0 AF1 AF2 AF3 ALT4	GPIO[108] E0UC[26] SOUT_4 — TXD[2]	SIUL eMIOS_0 DSPI_4 — FEC	I/O I/O O — O	M/S	Tristate	116	140	F15	F15
PG[13]	PCR[109]	AF0 AF1 AF2 AF3 ALT4	GPIO[109] E0UC[27] SCK_4 — TXD[3]	SIUL eMIOS_0 DSPI_4 — FEC	I/O I/O I/O — O	M/S	Tristate	115	139	F16	F16
PG[14]	PCR[110]	AF0 AF1 AF2 AF3 —	GPIO[110] E1UC[0] LIN8TX — SIN_6	SIUL eMIOS_1 LINFlexD_8 — DSPI_6	I/O I/O O — I	S	Tristate	134	158	C13	C13
PG[15]	PCR[111]	AF0 AF1 AF2 AF3 —	GPIO[111] E1UC[1] SOUT_6 — LIN8RX	SIUL eMIOS_1 DSPI_6 — LINFlexD_8	I/O I/O O — I	M/S	Tristate	135	159	D13	D13
PH[0]	PCR[112]	AF0 AF1 AF2 AF3 ALT4 —	GPIO[112] E1UC[2] — — TXD[1] SIN_1	SIUL eMIOS_1 — — FEC DSPI_1	I/O I/O — — O I	M/S	Tristate	117	141	E15	E15
PH[1]	PCR[113]	AF0 AF1 AF2 AF3 ALT4	GPIO[113] E1UC[3] SOUT_1 — TXD[0]	SIUL eMIOS_1 DSPI_1 — FEC	I/O I/O O — O	M/S	Tristate	118	142	F13	F13
PH[2]	PCR[114]	AF0 AF1 AF2 AF3 ALT4	GPIO[114] E1UC[4] SCK_1 — TX_EN	SIUL eMIOS_1 DSPI_1 — FEC	I/O I/O I/O — O	M/S	Tristate	119	143	D16	D16

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PH[3]	PCR[115]	AF0 AF1 AF2 AF3 ALT4	GPIO[115] E1UC[5] CS0_1 — TX_ER	SIUL eMIOS_1 DSPI_1 — FEC	I/O I/O I/O — O	M/S	Tristate	120	144	F14	F14
PH[4]	PCR[116]	AF0 AF1 AF2 AF3	GPIO[116] E1UC[6] SOUT_7 —	SIUL eMIOS_1 DSPI_7 —	I/O I/O O —	M/S	Tristate	162	186	D7	D7
PH[5]	PCR[117]	AF0 AF1 AF2 AF3 —	GPIO[117] E1UC[7] — — SIN_7	SIUL eMIOS_1 — — DSPI_7	I/O I/O — — I	S	Tristate	163	187	B7	B7
PH[6]	PCR[118]	AF0 AF1 AF2 AF3	GPIO[118] E1UC[8] SCK_7 MA[2]	SIUL eMIOS_1 DSPI_7 ADC_0	I/O I/O I/O O	M/S	Tristate	164	188	C7	C7
PH[7]	PCR[119]	AF0 AF1 AF2 AF3 ALT4	GPIO[119] E1UC[9] CS3_2 MA[1] CS0_7	SIUL eMIOS_1 DSPI_2 ADC_0 DSPI_7	I/O I/O O O I/O	M/S	Tristate	165	189	C6	C6
PH[8]	PCR[120]	AF0 AF1 AF2 AF3	GPIO[120] E1UC[10] CS2_2 MA[0]	SIUL eMIOS_1 DSPI_2 ADC_0	I/O I/O O O	M/S	Tristate	166	190	A6	A6
PH[9][6]	PCR[121]	AF0 AF1 AF2 AF3 —	GPIO[121] — — — TCK	SIUL — — — JTAGC	I/O — — — I	S	Input, weak pull-up	155	179	A11	A11

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PH[10] (6)	PCR[122]	AF0 AF1 AF2 AF3 —	GPIO[122] — — — TMS	SIUL — — — JTAGC	I/O — — — I	M/S	Input, weak pull-up	148	172	D10	D10
PH[11]	PCR[123]	AF0 AF1 AF2 AF3	GPIO[123] SOUT_3 CS0_4 E1UC[5]	SIUL DSPI_3 DSPI_4 eMIOS_1	I/O O I/O I/O	M/S	Tristate	140	164	A13	A13
PH[12]	PCR[124]	AF0 AF1 AF2 AF3	GPIO[124] SCK_3 CS1_4 E1UC[25]	SIUL DSPI_3 DSPI_4 eMIOS_1	I/O I/O O I/O	M/S	Tristate	141	165	B12	B12
PH[13]	PCR[125]	AF0 AF1 AF2 AF3	GPIO[125] SOUT_4 CS0_3 E1UC[26]	SIUL DSPI_4 DSPI_3 eMIOS_1	I/O O I/O I/O	M/S	Tristate	9	9	B1	B1
PH[14]	PCR[126]	AF0 AF1 AF2 AF3	GPIO[126] SCK_4 CS1_3 E1UC[27]	SIUL DSPI_4 DSPI_3 eMIOS_1	I/O I/O O I/O	M/S	Tristate	10	10	C1	C1
PH[15]	PCR[127]	AF0 AF1 AF2 AF3	GPIO[127] SOUT_5 — E1UC[17]	SIUL DSPI_5 — eMIOS_1	I/O O — I/O	M/S	Tristate	8	8	E3	E3
PI[0]	PCR[128]	AF0 AF1 AF2 AF3	GPIO[128] E0UC[28] LIN8TX —	SIUL eMIOS_0 LINFlexD_8 —	I/O I/O O —	S	Tristate	172	196	C5	C5
PI[1]	PCR[129]	AF0 AF1 AF2 AF3 — —	GPIO[129] E0UC[29] — — WKPU[24] LIN8RX	SIUL eMIOS_0 — — WKPU LINFlexD_8	I/O I/O — — I I	S	Tristate	171	195	A4	A4

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PI[2]	PCR[130]	AF0 AF1 AF2 AF3	GPIO[130] E0UC[30] LIN9TX —	SIUL eMIOS_0 LINFlexD_9 —	I/O I/O O —	S	Tristate	170	194	D6	D6
PI[3]	PCR[131]	AF0 AF1 AF2 AF3 — —	GPIO[131] E0UC[31] — — WKPU[23] LIN9RX	SIUL eMIOS_0 — — WKPU LINFlexD_9	I/O I/O — — I I	S	Tristate	169	193	B5	B5
PI[4]	PCR[132]	AF0 AF1 AF2 AF3	GPIO[132] E1UC[28] SOUT_4 —	SIUL eMIOS_1 DSPI_4 —	I/O I/O O —	M/S	Tristate	143	167	A12	A12
PI[5]	PCR[133]	AF0 AF1 AF2 AF3 ALT4	GPIO[133] E1UC[29] SCK_4 CS2_5 CS2_6	SIUL eMIOS_1 DSPI_4 DSPI_5 DSPI_6	I/O I/O I/O O O	M/S	Tristate	142	166	D12	D12
PI[6]	PCR[134]	AF0 AF1 AF2 AF3 ALT4	GPIO[134] E1UC[30] CS0_4 CS0_5 CS0_6	SIUL eMIOS_1 DSPI_4 DSPI_5 DSPI_6	I/O I/O I/O I/O I/O	S	Tristate	11	11	D2	D2
PI[7]	PCR[135]	AF0 AF1 AF2 AF3 ALT4	GPIO[135] E1UC[31] CS1_4 CS1_5 CS1_6	SIUL eMIOS_1 DSPI_4 DSPI_5 DSPI_6	I/O I/O O O O	S	Tristate	12	12	E2	E2
PI[8]	PCR[136]	AF0 AF1 AF2 AF3 —	GPIO[136] — — — ADC0_S[16]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	108	130	J14	J14

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PI[9]	PCR[137]	AF0 AF1 AF2 AF3 —	GPIO[137] — — — ADC0_S[17]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	—	131	J15	J15
PI[10]	PCR[138]	AF0 AF1 AF2 AF3 —	GPIO[138] — — — ADC0_S[18]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	—	134	J16	J16
PI[11]	PCR[139]	AF0 AF1 AF2 AF3 — —	GPIO[139] — — — ADC0_S[19] SIN_3	SIUL — — — ADC_0 DSPI_3	I/O — — — I I	S	Tristate	111	135	H16	H16
PI[12]	PCR[140]	AF0 AF1 AF2 AF3 —	GPIO[140] CS0_3 CS0_2 — ADC0_S[20]	SIUL DSPI_3 DSPI_2 — ADC_0	I/O I/O I/O — I	S	Tristate	112	136	G15	G15
PI[13]	PCR[141]	AF0 AF1 AF2 AF3 —	GPIO[141] CS1_3 CS1_2 — ADC0_S[21]	SIUL DSPI_3 DSPI_2 — ADC_0	I/O O O — I	S	Tristate	113	137	G14	G14
PI[14]	PCR[142]	AF0 AF1 AF2 AF3 — —	GPIO[142] — — — ADC0_S[22] SIN_4	SIUL — — — ADC_0 DSPI_4	I/O — — — I I	S	Tristate	76	92	T12	T12

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PI[15]	PCR[143]	AF0 AF1 AF2 AF3 —	GPIO[143] CS0_4 CS2_2 — ADC0_S[23]	SIUL DSPI_4 DSPI_2 — ADC_0	I/O I/O O — I	S	Tristate	75	91	P11	P11
PJ[0]	PCR[144]	AF0 AF1 AF2 AF3 —	GPIO[144] CS1_4 CS3_2 — ADC0_S[24]	SIUL DSPI_4 DSPI_2 — ADC_0	I/O O O — I	S	Tristate	74	90	R11	R11
PJ[1]	PCR[145]	AF0 AF1 AF2 AF3 — —	GPIO[145] — — — ADC0_S[25] SIN_5	SIUL — — —— ADC_0 DSPI_5	I/O — — — I I	S	Tristate	73	89	N10	N10
PJ[2]	PCR[146]	AF0 AF1 AF2 AF3 —	GPIO[146] CS0_5 CS0_6 CS0_7 ADC0_S[26]	SIUL DSPI_5 DSPI_6 DSPI_7 ADC_0	I/O I/O I/O I/O I	S	Tristate	72	88	R10	R10
PJ[3]	PCR[147]	AF0 AF1 AF2 AF3 —	GPIO[147] CS1_5 CS1_6 CS1_7 ADC0_S[27]	SIUL DSPI_5 DSPI_6 DSPI_7 ADC_0	I/O O O O I	S	Tristate	71	87	P10	P10
PJ[4]	PCR[148]	AF0 AF1 AF2 AF3	GPIO[148] SCK_5 E1UC[18] —	SIUL DSPI_5 eMIOS_1 —	I/O I/O I/O —	M/S	Tristate	5	5	D3	D3
PJ[5]	PCR[149]	AF0 AF1 AF2 AF3 —	GPIO[149] — — — ADC0_S[28]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	—	113	N12	N12

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PJ[6]	PCR[150]	AF0 AF1 AF2 AF3 —	GPIO[150] — — — ADC0_S[29]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	—	112	N15	N15
PJ[7]	PCR[151]	AF0 AF1 AF2 AF3 —	GPIO[151] — — — ADC0_S[30]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	—	111	P16	P16
PJ[8]	PCR[152]	AF0 AF1 AF2 AF3 —	GPIO[152] — — — ADC0_S[31]	SIUL — — — ADC_0	I/O — — — I	S	Tristate	—	110	P15	P15
PJ[9]	PCR[153]	AF0 AF1 AF2 AF3 —	GPIO[153] — — — ADC1_S[8]	SIUL — — — ADC_1	I/O — — — I	S	Tristate	—	68	P5	P5
PJ[10]	PCR[154]	AF0 AF1 AF2 AF3 —	GPIO[154] — — — ADC1_S[9]	SIUL — — — ADC_1	I/O — — — I	S	Tristate	—	67	T5	T5
PJ[11]	PCR[155]	AF0 AF1 AF2 AF3 —	GPIO[155] — — — ADC1_S[10]	SIUL — — — ADC_1	I/O — — — I	S	Tristate	—	60	R3	R3
PJ[12]	PCR[156]	AF0 AF1 AF2 AF3 —	GPIO[156] — — — ADC1_S[11]	SIUL — — — ADC_1	I/O — — — I	S	Tristate	—	59	T1	T1

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PJ[13]	PCR[157]	AF0 AF1 AF2 AF3 — — — —	GPIO[157] — CS1_7 — CAN4RX ADC1_S[12] CAN1RX WKPU[31]	SIUL — DSPI_7 — FlexCAN_4 ADC_1 FlexCAN_1 WKPU	I/O — O — I I I I	S	Tristate	—	65	N5	N5
PJ[14]	PCR[158]	AF0 AF1 AF2 AF3	GPIO[158] CAN1TX CAN4TX CS2_7	SIUL FlexCAN_1 FlexCAN_4 DSPI_7	I/O O O O	M/S	Tristate	—	64	T4	T4
PJ[15]	PCR[159]	AF0 AF1 AF2 AF3 —	GPIO[159] — CS1_6 — CAN1RX	SIUL — DSPI_6 — FlexCAN_1	I/O — O — I	M/S	Tristate	—	63	R4	R4
PK[0]	PCR[160]	AF0 AF1 AF2 AF3	GPIO[160] CAN1TX CS2_6 —	SIUL FlexCAN_1 DSPI_6 —	I/O O O —	M/S	Tristate	—	62	T3	T3
PK[1]	PCR[161]	AF0 AF1 AF2 AF3 —	GPIO[161] CS3_6 — — CAN4RX	SIUL DSPI_6 — — FlexCAN_4	I/O O — — I	M/S	Tristate	—	41	H4	H4
PK[2]	PCR[162]	AF0 AF1 AF2 AF3	GPIO[162] CAN4TX — —	SIUL FlexCAN_4 — —	I/O O — —	M/S	Tristate	—	42	L4	L4
PK[3]	PCR[163]	AF0 AF1 AF2 AF3 — —	GPIO[163] E1UC[0] — — CAN5RX LIN8RX	SIUL eMIOS_1 — — FlexCAN_5 LINFlexD_8	I/O I/O — — I I	M/S	Tristate	—	43	N1	N1

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PK[4]	PCR[164]	AF0 AF1 AF2 AF3	GPIO[164] LIN8TX CAN5TX E1UC[1]	SIUL LINFlexD_8 FlexCAN_5 eMIOS_1	I/O O O I/O	M/S	Tristate	—	44	M3	M3
PK[5]	PCR[165]	AF0 AF1 AF2 AF3 — —	GPIO[165] — — — CAN2RX LIN2RX	SIUL — — — FlexCAN_2 LINFlexD_2	I/O — — — I I	M/S	Tristate	—	45	—	M5
PK[6]	PCR[166]	AF0 AF1 AF2 AF3	GPIO[166] CAN2TX LIN2TX —	SIUL FlexCAN_2 LINFlexD_2 —	I/O O O —	M/S	Tristate	—	46	—	M6
PK[7]	PCR[167]	AF0 AF1 AF2 AF3 — —	GPIO[167] — — — CAN3RX LIN3RX	SIUL — — — FlexCAN_3 LINFlexD_3	I/O — — — I I	M/S	Tristate	—	47	—	M7
PK[8]	PCR[168]	AF0 AF1 AF2 AF3	GPIO[168] CAN3TX LIN3TX —	SIUL FlexCAN_3 LINFlexD_3 —	I/O O O —	M/S	Tristate	—	48	—	M8
PK[9]	PCR[169]	AF0 AF1 AF2 AF3 —	GPIO[169] — — — SIN_4	SIUL — — — DSPI_4	I/O — — — I	M/S	Tristate	—	197	—	E8
PK[10]	PCR[170]	AF0 AF1 AF2 AF3	GPIO[170] SOUT_4 — —	SIUL DSPI_4 — —	I/O O — —	M/S	Tristate	—	198	—	E7

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PK[11]	PCR[171]	AF0 AF1 AF2 AF3	GPIO[171] SCK_4 — —	SIUL DSPI_4 — —	I/O I/O — —	M/S	Tristate	—	199	—	F8
PK[12]	PCR[172]	AF0 AF1 AF2 AF3	GPIO[172] CS0_4 — —	SIUL DSPI_4 — —	I/O I/O — —	M/S	Tristate	—	200	—	G12
PK[13]	PCR[173]	AF0 AF1 AF2 AF3 —	GPIO[173] CS3_6 CS2_7 SCK_1 CAN3RX	SIUL DSPI_6 DSPI_7 DSPI_1 FlexCAN_3	I/O O O I/O I	M/S	Tristate	—	201	—	H12
PK[14]	PCR[174]	AF0 AF1 AF2 AF3	GPIO[174] CAN3TX CS3_7 CS0_1	SIUL FlexCAN_3 DSPI_7 DSPI_1	I/O O O I/O	M/S	Tristate	—	202	—	J12
PK[15]	PCR[175]	AF0 AF1 AF2 AF3 — —	GPIO[175] — — — SIN_1 SIN_7	SIUL — — — DSPI_1 DSPI_7	I/O — — — I I	M/S	Tristate	—	203	D5	D5
PL[0]	PCR[176]	AF0 AF1 AF2 AF3	GPIO[176] SOUT_1 SOUT_7 —	SIUL DSPI_1 DSPI_7 —	I/O O O —	M/S	Tristate	—	204	C4	C4
PL[1]	PCR[177]	AF0 AF1 AF2 AF3	GPIO[177] — — —	SIUL — — —	I/O — — —	M/S	Tristate	—	—	—	F7
PL[2]	PCR[178][7]	AF0 AF1 AF2 AF3	GPIO[178] — MDO0[8] —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	F5

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PL[3]	PCR[179]	AF0 AF1 AF2 AF3	GPIO[179] — MDO1 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	G5
PL[4]	PCR[180]	AF0 AF1 AF2 AF3	GPIO[180] — MDO2 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	H5
PL[5]	PCR[181]	AF0 AF1 AF2 AF3	GPIO[181] — MDO3 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	J5
PL[6]	PCR[182]	AF0 AF1 AF2 AF3	GPIO[182] — MDO4 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	K5
PL[7]	PCR[183]	AF0 AF1 AF2 AF3	GPIO[183] — MDO5 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	L5
PL[8]	PCR[184]	AF0 AF1 AF2 AF3 —	GPIO[184] — — — EVTI	SIUL — — — Nexus	I/O — — — I	S	Pull-up	—	—	—	M9
PL[9]	PCR[185]	AF0 AF1 AF2 AF3	GPIO[185] — MSEO —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	M10
PL[10]	PCR[186]	AF0 AF1 AF2 AF3	GPIO[186] — MCKO —	SIUL — Nexus —	I/O — O —	F/S	Tristate	—	—	—	M11

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PL[11]	PCR[187]	AF0 AF1 AF2 AF3	GPIO[187] — — —	SIUL — — —	I/O — — —	M/S	Tristate	—	—	—	M12
PL[12]	PCR[188]	AF0 AF1 AF2 AF3	GPIO[188] — EVTO —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	F11
PL[13]	PCR[189]	AF0 AF1 AF2 AF3	GPIO[189] — MDO6 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	F10
PL[14]	PCR[190]	AF0 AF1 AF2 AF3	GPIO[190] — MDO7 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	E12
PL[15]	PCR[191]	AF0 AF1 AF2 AF3	GPIO[191] — MDO8 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	E11
PM[0]	PCR[192]	AF0 AF1 AF2 AF3	GPIO[192] — MDO9 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	E10
PM[1]	PCR[193]	AF0 AF1 AF2 AF3	GPIO[193] — MDO10 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	E9
PM[2]	PCR[194]	AF0 AF1 AF2 AF3	GPIO[194] — MDO11 —	SIUL — Nexus —	I/O — O —	M/S	Tristate	—	—	—	F12

Table 10. Functional port pin descriptions (continued)

Port pin	PCR	Alternate function[1]	Function	Peripheral	I/O direction[2]	Pad type	RESET config.	Pin number			
								LQFP 176	LQFP 208	LBGA208	LBGA256
PM[3]	PCR[195]	AF0 AF1 AF2 AF3	GPIO[195] — — —	SIUL — — —	I/O — — —	M/S	Tristate	—	—	—	K12
PM[4]	PCR[196]	AF0 AF1 AF2 AF3	GPIO[196] — — —	SIUL — — —	I/O — — —	M/S	Tristate	—	—	—	L12
PM[5]	PCR[197]	AF0 AF1 AF2 AF3	GPIO[197] — — —	SIUL — — —	I/O — — —	M/S	Tristate	—	—	—	F9
PM[6]	PCR[198]	AF0 AF1 AF2 AF3	GPIO[198] — — —	SIUL — — —	I/O — — —	M/S	Tristate	—	—	—	F6

1. Alternate functions are chosen by setting the values of the PCR.PA bitfields inside the SIUL module. PCR.PA = 000 → AF0; PCR.PA = 001 → AF1; PCR.PA = 010 → AF2; PCR.PA = 011 → AF3; PCR.PA = 100 → ALT4. 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. NMI[0] and NMI[1] have a higher priority than alternate functions. When NMI is selected, the PCR.PA field is ignored.
4. SXOSC's OSC32k_XTAL and OSC32k_EXTAL pins are shared with GPIO functionality. When used as crystal pins, other functionality of the pin cannot be used and it should be ensured that application never programs OBE and PUE bit of the corresponding PCR to "1".
5. If you want to use OSC32K functionality through PB[8] and PB[9], you must ensure that PB[10] is static in nature as PB[10] can induce coupling on PB[9] and disturb oscillator frequency.
6. 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).
   It is up to the user to configure these pins as GPIO when needed.
7. When MBIST is enabled to run (STCU Enable = 1), the application must not drive or tie PAD[178] (MDO[0]) to 0 V before the device exits reset (external reset is removed) as the pad is internally driven to 1 to indicate MBIST operation. When MBIST is not enabled (STCU Enable = 0), there are no restriction as the device does not internally drive the pad.
8. These pins can be configured as Nexus pins during reset by the debugger writing to the Nexus Development Interface "Port Control Register" rather than the SIUL. Specifically, the debugger can enable the MDO[7:0], MSEO, and MCKO ports by programming NDI (PCR[MCKO_EN] or PCR[PSTAT_EN]). MDO[8:11] ports can be enabled by programming NDI ((PCR[MCKO_EN] and PCR[FPM]) or PCR[PSTAT_EN]).

4.8 Nexus 3+ pins

In the LBGA256 package, seventeen pins are dedicated for Nexus (see *Table 11*). These pins are available only in the 256-pin BGA package.

Table 11. Nexus 3+ pin descriptions

Port pin	Function	I/O direction	Pad type	Function after reset	Pin number
					LBGA256
MCKO	Message clock out	O	F/M	—	M11
MDO0	Message data out 0	O	M/S	—	F5
MDO1	Message data out 1	O	M/S	—	G5
MDO2	Message data out 2	O	M/S	—	H5
MDO3	Message data out 3	O	M/S	—	J5
MDO4	Message data out 4	O	M/S	—	K5
MDO5	Message data out 5	O	M/S	—	L5
MDO6	Message data out 6	O	M/S	—	F10
MDO7	Message data out 7	O	M/S	—	E12
MDO8	Message data out 8	O	M/S	—	E11
MDO9	Message data out 9	O	M/S	—	E10
MDO10	Message data out 10	O	M/S	—	E9
MDO11	Message data out 11	O	M/S	—	F12
EVTI	Event in	I	S	Pull-up	M9
EVTO	Event out	O	M/S	—	F11
MSEO0	Message start/end out	O	M/S	—	M10

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 *Chapter 35: 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.

The CSE and MBIST boot behavior is not covered here, but is described in the respective reference manual chapters.

There are two 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 LINFlexD (RS232):

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

Table 12 describes the configuration options.

Table 12. Boot mode selection

Mode	FAB pin (PA[9])	ABS pin (PA[8])
Flash memory boot (default mode)	0	X
Serial boot (LINFlexD)	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*.

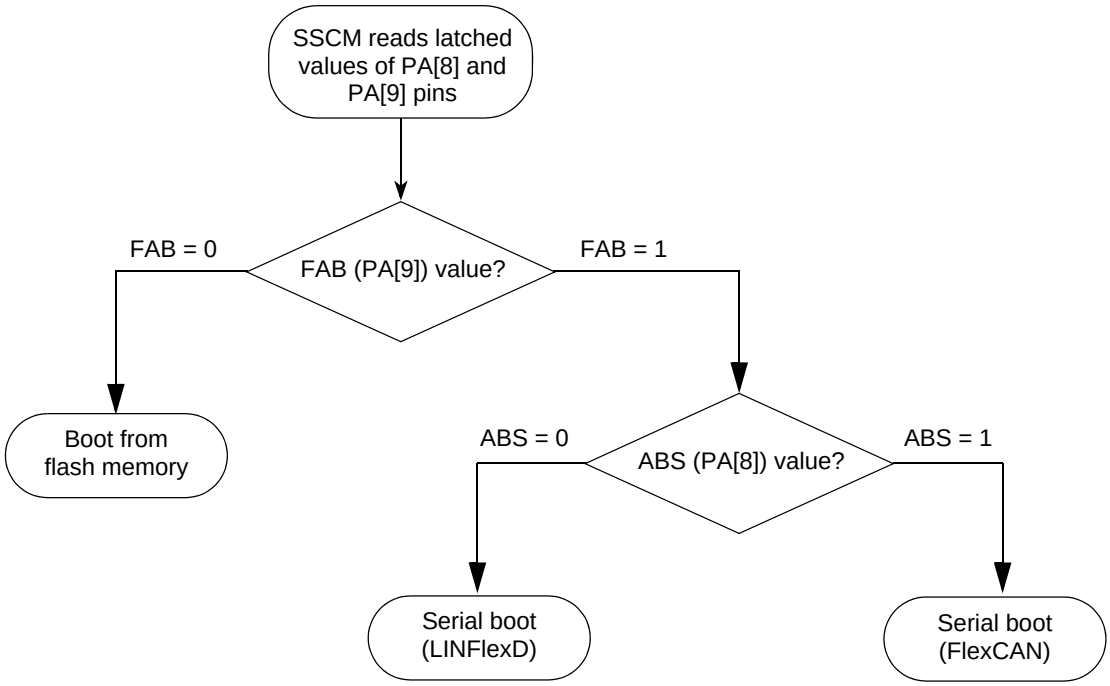


Figure 7. Boot mode selection

5.1.1 Flash memory boot

In order to successfully boot from flash memory, you must program two 32-bit fields into one of five 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
  - A VLE bit which configures the initial MMU entry to either Power Architecture Book VLE or Power Architecture Book III-E as described later on in this chapter
- 32-bit reset vector (this is the start address of the user code and for the CSE BOOT_MAC calculation)

The boot sector also contains a 32-bit field containing the size of the block of data to be checked by the CSE during a secure boot. See the CSE chapter for more details. Application code can be programmed from offset address 0x000C.

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

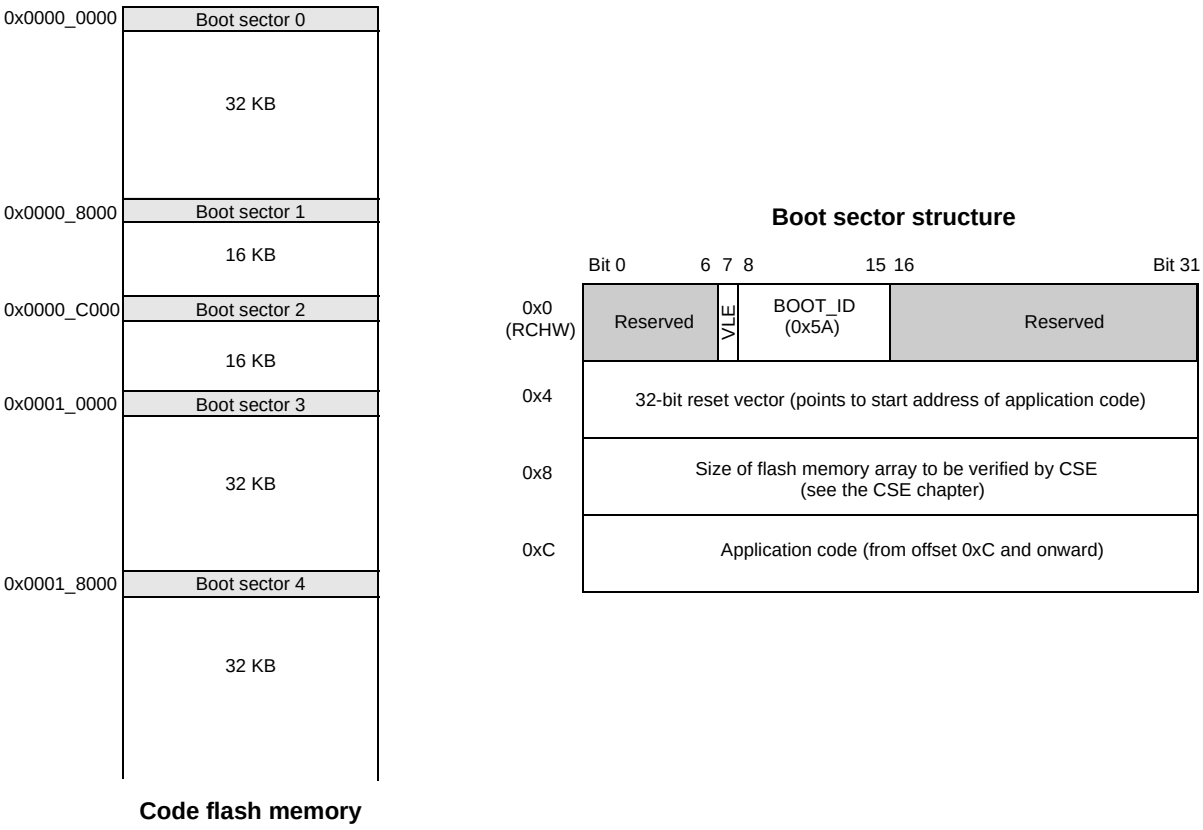


Figure 8. Boot sector structure

The RCHW fields are described in *Table 13*.

Table 13. RCHW field descriptions

Field	Description
VLE	VLE Bit 0 MMU TLB Entry 0 is configured for Power Architecture Book III-E. 1 MMU TLB Entry 0 is configured for Power Architecture Book VLE.
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 VLE bit and the boot vector address (as well as the CSE block size). If a valid BOOT_ID is not found, the SSCM starts the process of putting the microcontroller into static mode.

In order for the e200z4d core to be able to access memory, a valid MMU TLB entry has to be created. The SSCM does this automatically by reading the reset vector and modifying TLB entry 0 to create a 4 KB page containing the reset vector address. The MMU VLE bit is set depending on the status of the VLE bit within the RCHW. The 4 KB MMU page must be

4 KB aligned. This means that the most efficient place to put the application code is immediately after the boot sector. The 4 KB block provides sufficient space to:

- Add MMU entries for SRAM and peripherals
- Perform standard system initialisation tasks (initialise the SRAM, setup stack, copy constant data)
- Transfer execution to RAM, re-define the flash memory MMU entry and transfer execution back to flash memory.

Finally, the SSCM sets the e200z4d core instruction pointer to the reset vector address and starts the core running.

5.1.1.1 Static mode

If no valid BOOT_ID within the RCHW was found, the SSCM creates a 4 KB MMU page at the start of the BAM with the VLE bit set (the BAM is VLE code). The SSCM then 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 Software Watchdog Timer (SWT) is enabled.
- The core executes the "wait" instruction which halts the core.

After the microcontroller enters static mode, the SWT periodically resets the core (approximately every 10 ms) to re-attempt a boot from flash memory.

The sequence is illustrated in *Figure 9*.

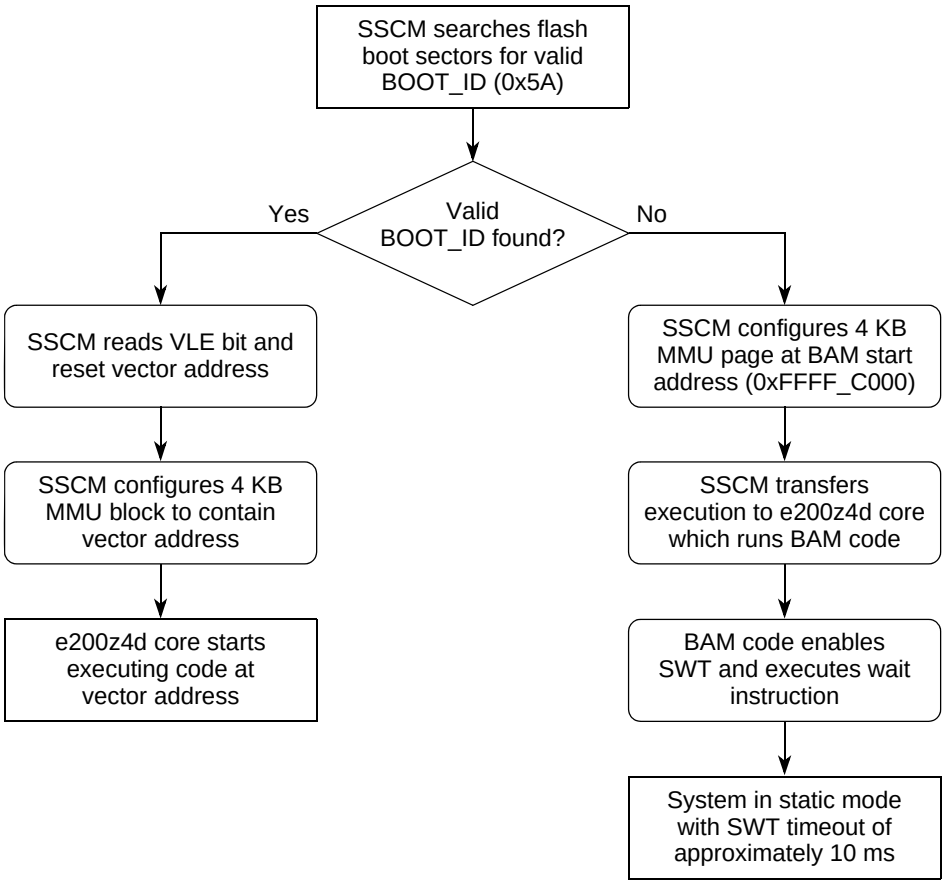


Figure 9. Flash memory boot mode sequence

5.1.1.2 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 LINFlexD (RS232). After the SSCM has detected that serial boot mode has been requested, a 4 KB MMU page is created at the start of the BAM and 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 626* for a full description):

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

5.1.3.1 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 14*:

Table 14. 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 LINFlexD 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.

5.1.3.2 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.

5.1.3.3 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 15 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 15. 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 15. 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 OR NVSCC0 != NVSCC1		Disabled	Disabled	Public (0xFEED_FACE_CAFE_BEEF)	

= 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.

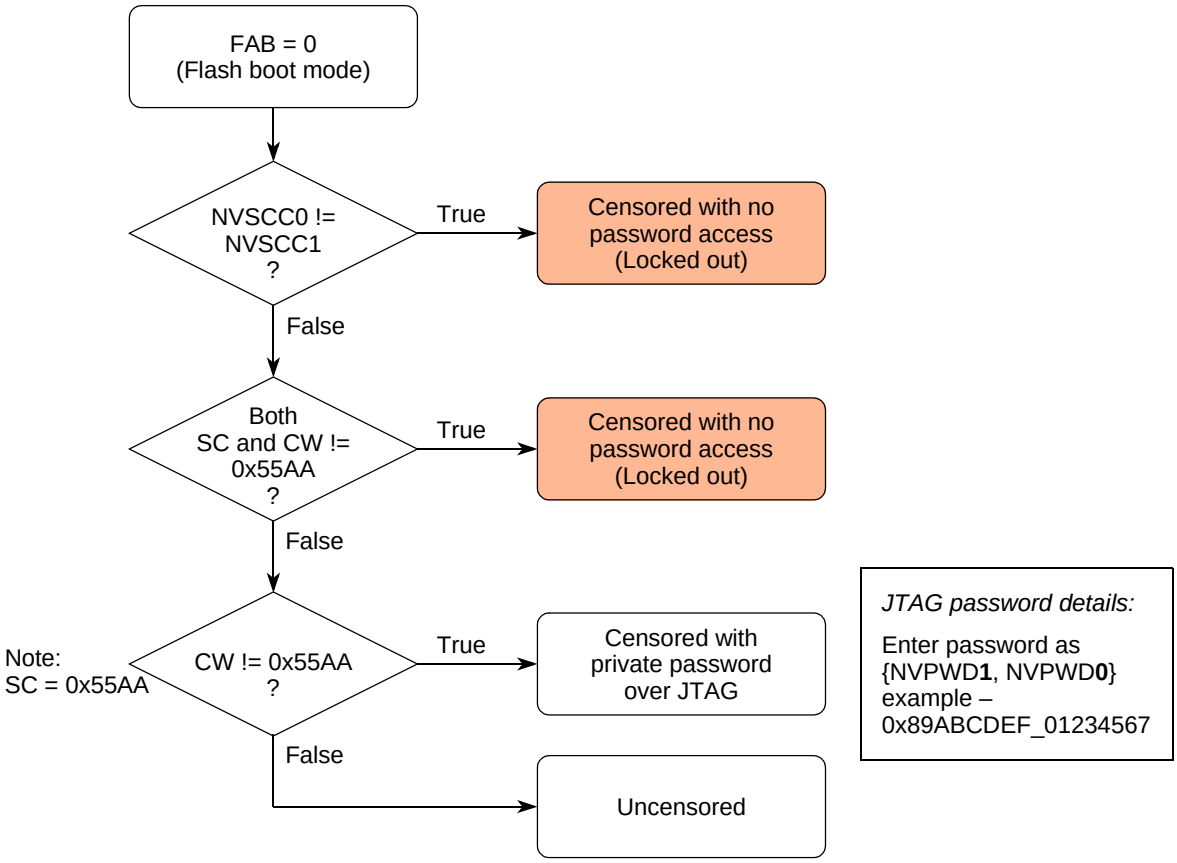


Figure 10. Censorship control in flash memory boot mode

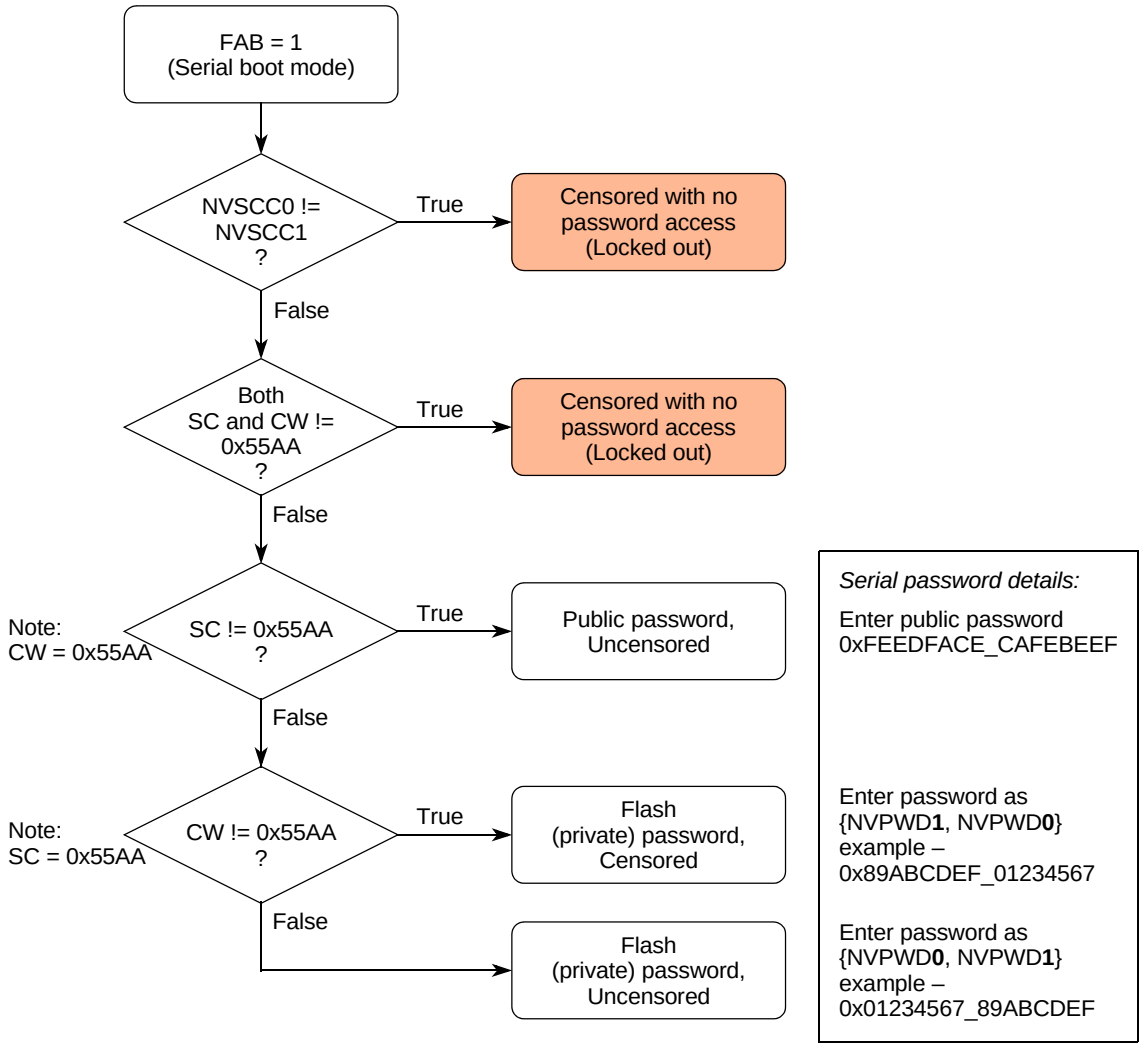


Figure 11. Censorship control in serial boot mode

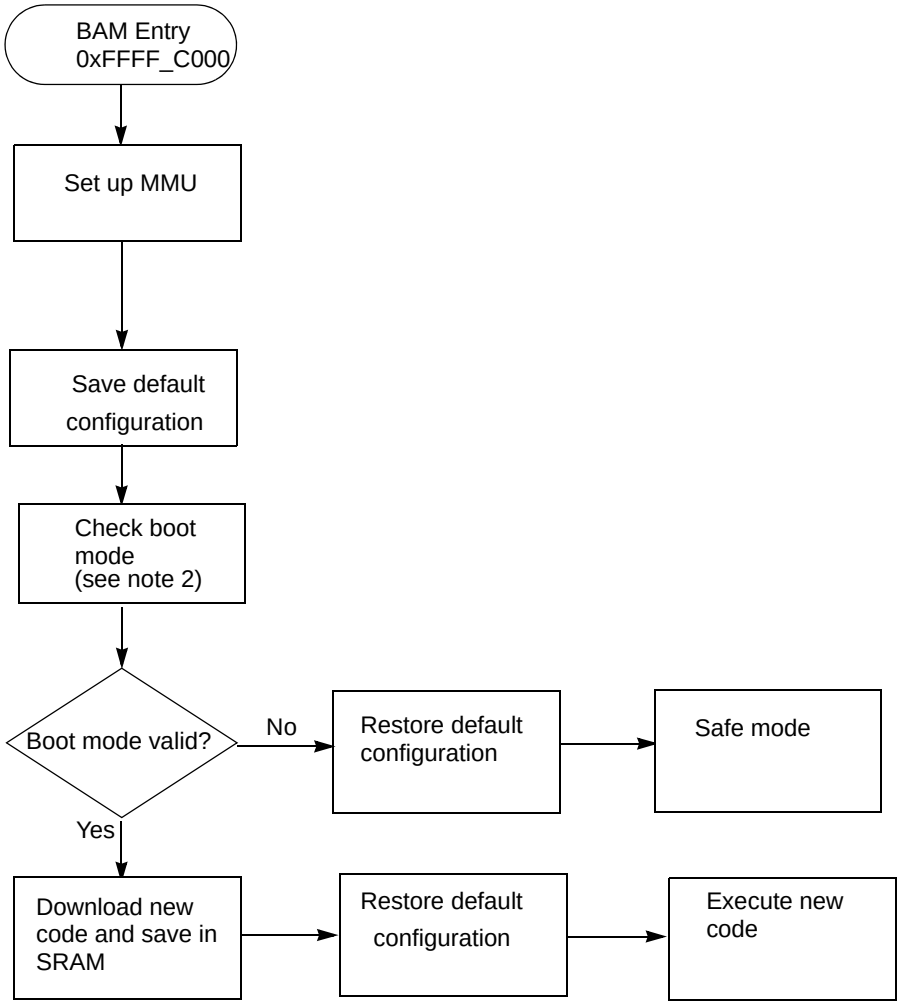
5.2 Boot Assist Module (BAM)

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

- Manages the serial download (FlexCAN or LINFlexD 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.



1) Boot core is determined by Processor Version Register (PVR)
2) The Boot Mode selection is done reading the SSCM_STATUS register

Figure 12. BAM logic flow

The first action is to set up the MMU for the e200z4d core to allow access to the SRAM.

Next, 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 16*). This field is only updated during reset.

There are two 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 the SWT is enabled and a wait instruction is executed to halt the core.

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

Table 16. SSCM_STATUS[BMODE] values as used by BAM

BMODE value	Corresponding boot mode
000	Reserved
001	FlexCAN_0 serial boot loader
010	LINFlexD_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.

5.2.1.1 BAM resources

The BAM uses/initializes the following MCU resources:

- MMU to initalize access to resources. This is only initialized if e200z4d is the primary core.
- MC_ME and MC_CGM to initialize mode and clock sources
- FlexCAN_0, LINFlexD_0 and the respective I/O pins when performing serial boot mode
- SSCM during password check
- SSCM to check the boot mode (see *Table 16*)
- 4–40 MHz external crystal oscillator

The system clock is selected directly from the 4–40 MHz external crystal oscillator. Thus, the external oscillator frequency defines the baud rates used for serial download (see *Table 17*).

Table 17. Serial boot mode – baud rates

FXOSC frequency (MHz)	LINFlexD 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
20	24000	500K
40	48000	1M

5.2.1.2 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.
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.

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.

5.2.1.3 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 5.3.4.1: System Status Register (SSCM_STATUS)*) as shown in *Table 18*.

Table 18. 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**}.

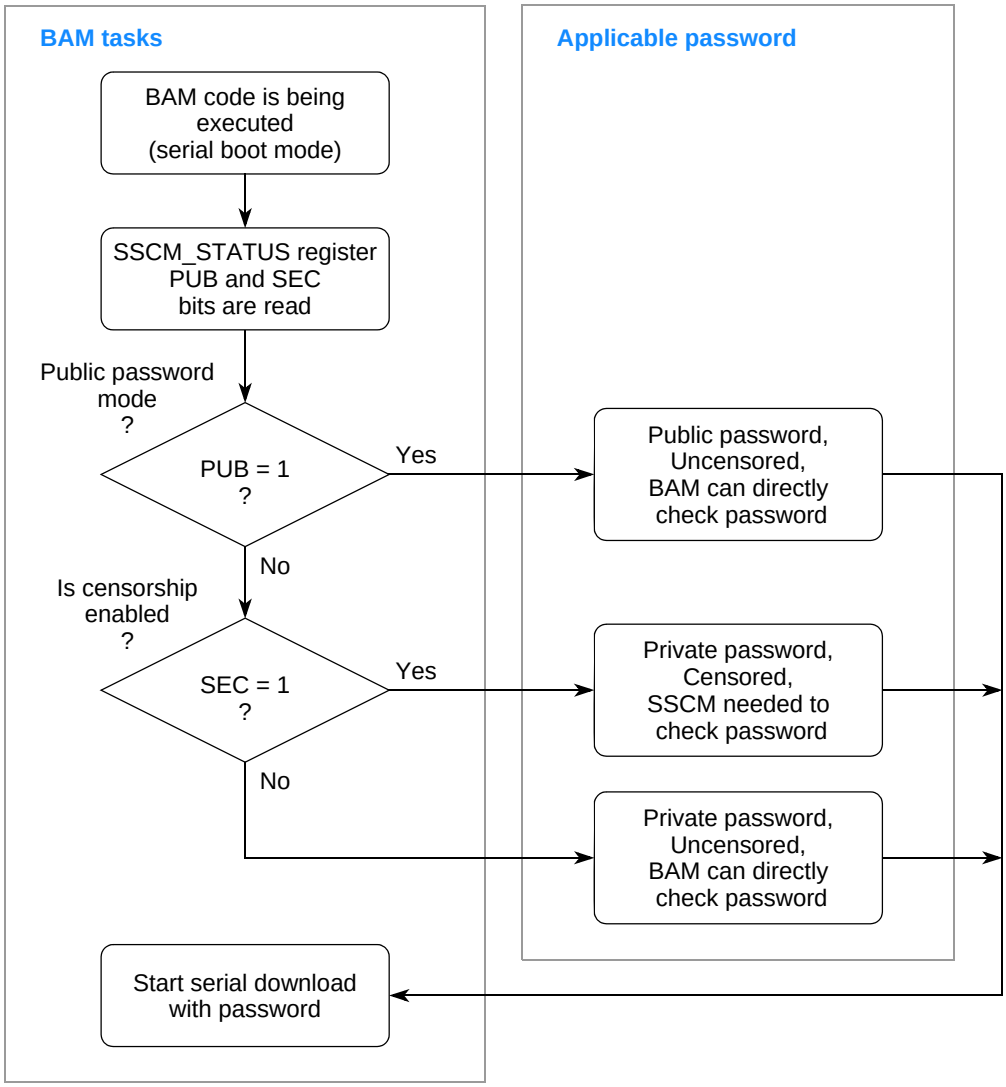


Figure 13. BAM censorship mode detection

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.

5.2.1.3.1 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.

5.2.1.3.2 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.

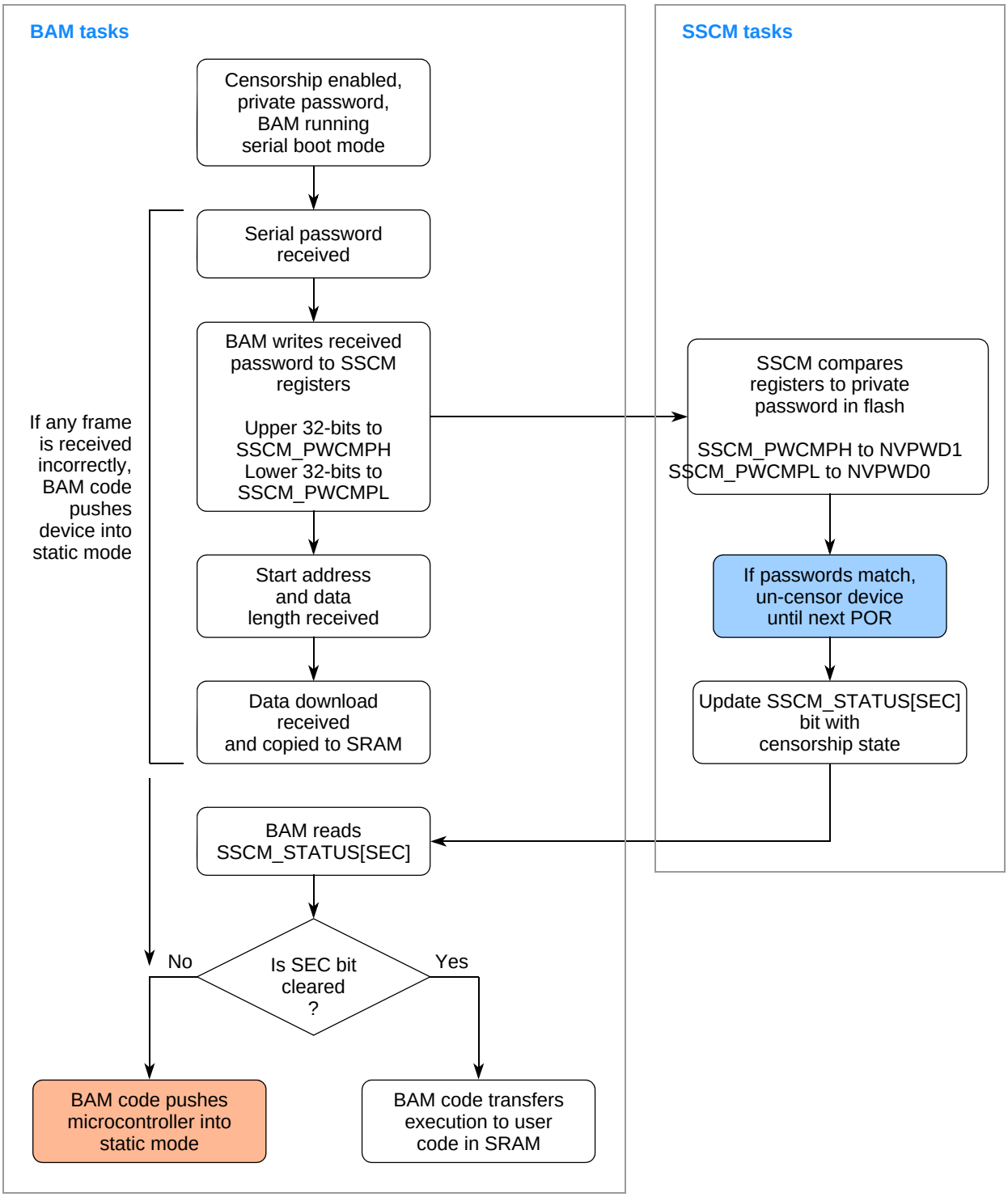


Figure 14. BAM serial boot mode flow for censorship enabled and private password

With LINFlexD, 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.

5.2.1.4 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

START_ADDRESS[31:16]

START_ADDRESS[15:0]

VLE	CODE_LENGTH[30:16]

CODE_LENGTH[15:0]

The VLE bit (Variable Length Instruction) is used to indicate whether the code to be downloaded is Book VLE or Book III-E. This in turn will define how the MMU page is configured for the SRAM where the code is downloaded.

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 two 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.

Note: Start address should be 64-bit aligned, and the code length should be such that the end address comes to be 256-bit aligned, only if cache is enabled.

5.2.1.5 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.

Note: The BAM code initializes the SRAM area used for BAM operation (and ensures that received data is programmed to SRAM in 32-bit blocks) so no ECC errors are encountered.

Note: The rest of the SRAM area remains untouched and must be initialized by the user application code before use.

5.2.1.6 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.

5.2.2 LINFlexD (RS232) boot

5.2.2.1 Configuration

Boot according to the LINFlexD boot mode download protocol (see *Section 5.2.2.2: Protocol*) is performed by the LINFlexD_0 module in UART (RS232) mode. Pins used are:

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

Boot from LINFlexD uses the system clock driven by the 4–40 MHz fast external crystal oscillator (FXOSC).

The LINFlexD 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.

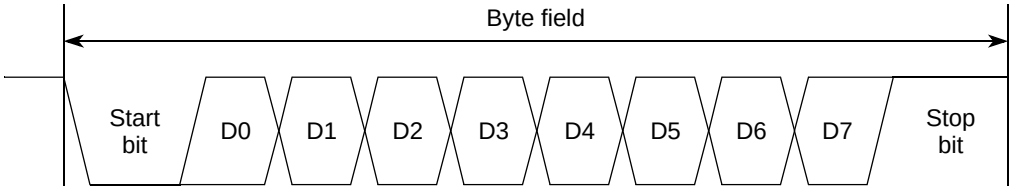


Figure 16. LINFlexD bit timing in UART mode

5.2.2.2 Protocol

Table 19 summarizes the protocol and BAM action during this boot mode.

Table 19. 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.

Table 19. UART boot mode download protocol

Protocol step	Host sent message	BAM response message	Action
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. Configure SRAM MMU page as either Book VLE or Book III-E based on setting of VLE bit.
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

5.2.3.1 Configuration

Boot according to the FlexCAN boot mode download protocol (see *Section 5.2.3.2: Protocol*) is performed by the FlexCAN_0 module. Pins used are:

- CAN0_TX mapped on PB[0]
- CAN0_RX mapped on PB[1]

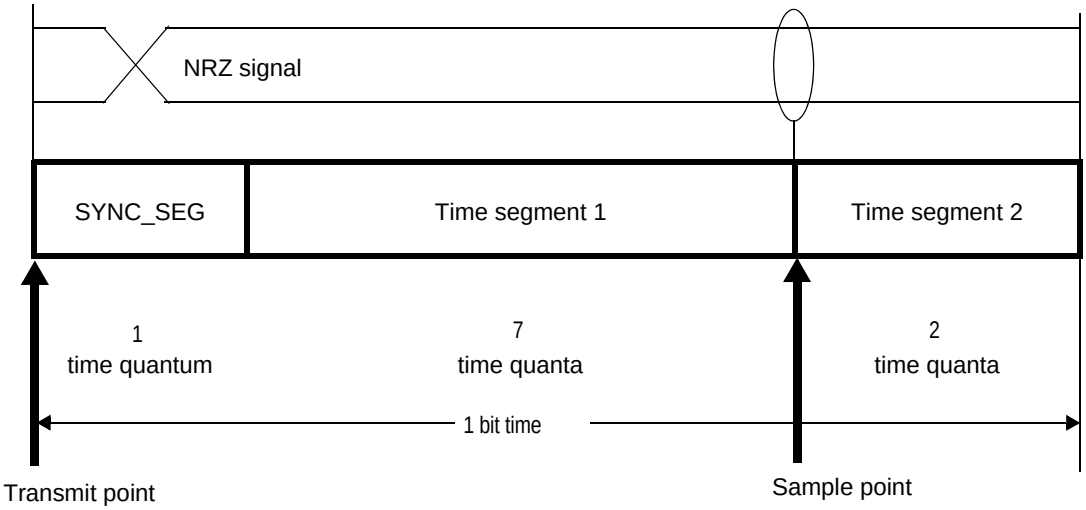
Note: When the serial download via FlexCAN is selected and the device is part of a CAN network, the serial download may stop unexpectedly if there is any other traffic on the network. To avoid this situation, ensure that no other CAN device on the network is active during the serial download process.

Boot from FlexCAN uses the system clock driven by the 4–40 MHz fast external crystal oscillator.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency/40 (see *Table 17* 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

5.2.3.2 Protocol

Table 20 summarizes the protocol and BAM action during this boot mode. All data are transmitted byte wise.

Table 20. 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. Configure SRAM MMU page as either Book VLE or Book III-E based on setting of VLE bit
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.

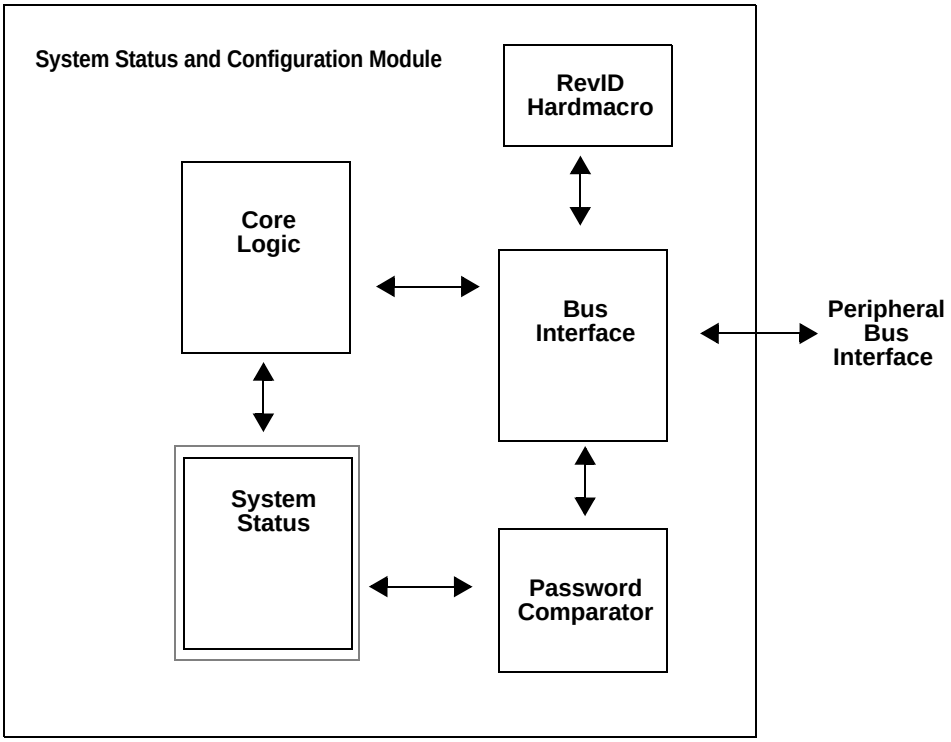


Figure 18. SSCM block diagram

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
- Read STCU test parameters from NVM / shadow flash memory and copy them into the STCU
- Read the CSE boot block size from the boot clock and issue the SECURE_BOOT command to CSE to start secure boot process

5.3.3 Modes of operation

The SSCM operates identically in all system modes.

5.3.4 Memory map and register description

Table 21 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 21. SSCM memory map

Address offset	Register	Location
0x00	System Status Register (SSCM_STATUS)	*on page 161*
0x02	System Memory Configuration Register (SSCM_MEMCONFIG)	*on page 163*
0x04	Reserved	
0x06	Error Configuration (SSCM_ERROR)	*on page 164*
0x08	Debug Status Port Register (SSCM_DEBUGPORT)	*on page 165*
0x08–0x0A	Reserved	
0x0C	Password Comparison Register High Word (SSCM_PWCMPH)	*on page 166*
0x10	Password Comparison Register Low Word (SSCM_PWCMPL)	*on page 166*
0x14	Reserved	
0x18	DPM Boot Register (SSCM_DPMBOOT)	*on page 167*
0x1C	DPM Boot Key Register (SSCM_DPMKEY)	*on page 168*
0x20	User Option Status Register (SSCM_UOPS)	*on page 169*
0x24	Reserved	
0x28	Processor Start Address register (SSCM_PSA)	*on page 170*
0x2C	Code Length Register (SSCM_CLEN)	*on page 170*

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.

5.3.4.1 System Status Register (SSCM_STATUS)

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

Offset:0x00 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	CER	0	Z4_NXEN	Z0_NXEN	PUB	SEC	0	BMODE			VLE	0	0	0	0
W		w1c														
Reset	0	0	0	0	0	0	0	0	0/1	0/1	0/1	0	0	0	0	0

Figure 19. System Status register (SSCM_STATUS)

Table 22. 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 23. SSCM_STATUS field descriptions

Field	Description
CER	Configuration Error. This field indicates that the SSCM has detected a configuration error during bootup. 0 No configuration problem detected by the SSCM. 1 Device configuration is not correct.
Z4_NXEN	e200z4d Nexus enabled
Z0_NXEN	e200z0h Nexus enabled
PUB	Public Serial Access Status. This bit indicates whether serial boot mode with public password is allowed. 0 Serial boot mode with private flash memory password is allowed 1 Serial boot mode with public password is allowed
SEC	Security Status. This bit reflects the current security state of the flash memory. 0 The flash memory is not secured. 1 The flash memory is secured.

Table 23. SSCM_STATUS field descriptions (continued)

Field	Description
BMODE	Device Boot Mode. 000 Reserved 001 FlexCAN_0 Serial Boot Loader 010 LINFlexD_0 Serial Boot Loader 011 Single Chip 100 Reserved 101 Reserved 110 Reserved 111 Reserved This field is only updated during reset.
VLE	Variable Length Instruction Mode. When booting from flash memory, this field indicates that the code stored there is using the VLE instruction set. The value of this bit is determined by the VLE field in the RCHW of the flash memory boot sector. 0 Code flash memory contains Book III-E code 1 Code flash memory contains Book VLE code

5.3.4.2 System Memory Configuration Register (SSCM_MEMCONFIG)

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

Figure 20. System Memory Configuration register (SSCM_MEMCONFIG)

Offset: Base + 0x0002 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	JPIN										ILVD	MREV				DVLD
W																
Reset	1(1)	0(1)	0(1)	1(1)	0(1)	0(1)	1(1)	0(1)	0(1)	0(1)	1(2)	0(2)	0(2)	0(2)	0(2)	1(2)

1. Reset value is device-specific.
2. Reset value depends on boot mode and security status.

Table 24. SSCM_MEMCONFIG field descriptions

Field	Description
JPIN	JTAG Part ID Number. JPIN reset value is 0x249.
ILVD	Instruction Flash Valid. This bit identifies whether or not the on-chip Instruction flash memory is accessible in the system memory map. The flash memory may not be accessible due to security limitations, or because there is no flash memory in the system. 0 Instruction flash memory is not accessible 1 Instruction flash memory is accessible

Table 24. SSCM_MEMCONFIG field descriptions (continued)

Field	Description
MREV	Minor Mask Revision. MREV reset value is 0x0.
DVLD	Data flash memory Valid. This bit identifies whether or not the on-chip data flash memory is visible in the system memory map. The flash memory may not be accessible due to security limitations, or because there is no flash memory in the system. 0 Data flash memory is not visible 1 Data flash memory is visible

Table 25. 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

5.3.4.3 Error Configuration (SSCM_ERROR)

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

Figure 21. Error Configuration (SSCM_ERROR)

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

Table 26. 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. 0 Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception 1 Illegal accesses to non-existing peripherals 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. 0 Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception 1 Illegal accesses to peripherals 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 27. SSCM_ERROR allowed register accesses

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

5.3.4.4 Debug Status Port Register (SSCM_DEBUGPORT)

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

Figure 22. Debug Status Port Register (SSCM_DEBUGPORT)

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

Table 28. 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 29* describes the functionality of the Debug Status Port in each mode.

Table 29. 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

Table 29. Debug status port modes (continued)

Pin (1)	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
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 29* equates to PC[2..9] (Pad 34..41).

Table 30. 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).

5.3.4.5 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.

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 23. Password Comparison Register High Word (SSCM_PWCMPH)

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

Figure 24. Password Comparison Register Low Word (SSCM_PWCMPL)

Table 31. Password Comparison Register field descriptions

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

Table 32. 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.

5.3.4.6 DPM Boot Register (SSCM_DPMBOOT)

This register is used in two functional use cases:

- After normal RESET BOOT – The register is used to wake up the e200z0h
- After STANDBY BOOT into RAM – The register is used to wake up the alternate core

Offset: 0x0018 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PBOOT[29:14]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PBOOT[13:0]														0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25. DPM Boot register (SSCM_DPMBOOT)

Table 33. SSCM_DPMBOOT field descriptions

Field	Description
PBOOT	"Determines the start address from which the non-running core will boot. From a POR, this will always refer to the e200z0h core. From STANDBY mode exit, this could refer to either the e200z0h core or the e200z4d core depending on which core is set to be active on STANDBY exit."

5.3.4.7 DPM Boot Key Register (SSCM_DPMKEY)

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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	KEY															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26. DPM Boot Key register (SSCM_DPMKEY)

Table 34. SSCM_DPMKEY field descriptions

Field	Description
KEY	Control key. This field is used to activate the e200z0h core The following sequence is required: - write to the SSCM_DPMBOOT register - write the value 0101101011110000 (0x5AF0) to the KEY field - write the value 1010010100001111 (0xA50F) to the KEY field After this the second core will start executing from the address specified in the SSCM_DPMBOOT register.

5.3.4.8 User Option Status Register (SSCM_UOPS)

Offset: 0x0020 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UOPT[31:16]															
W																
Reset	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)	0/1(1)

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

Figure 27. User Option Status register (SSCM_UOPS)

1. Default reset value is 0. It cannot be modified by software.

Table 35. SSCM_UOPS field descriptions

Field	Description
UOPT	Shows the values read from the User Option Bits location in the Flash.

5.3.4.9 Processor Start Address Register (SSCM_PSA)

Offset: 0x0028 Access: Read/Write

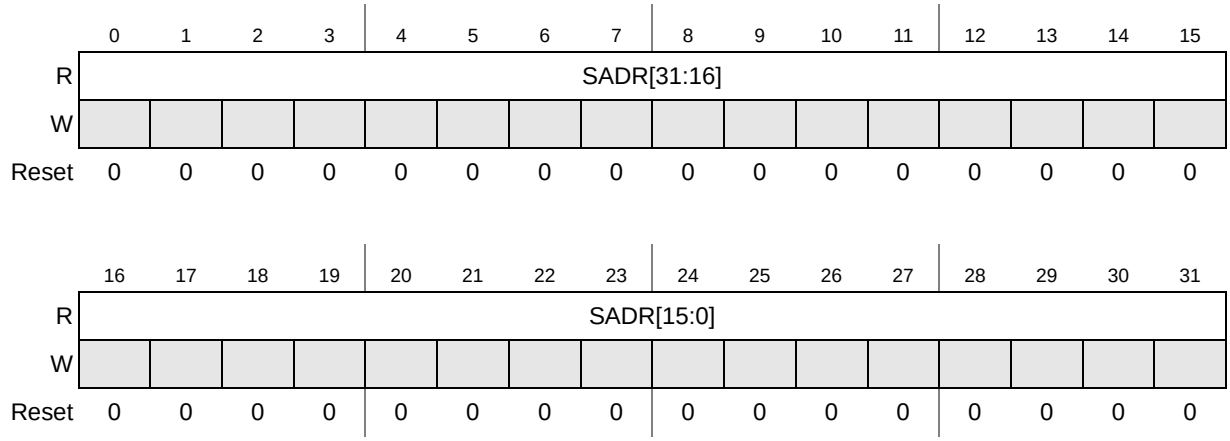


Figure 28. Processor Start Address register (SSCM_PSA)

Table 36. SSCM_PSA field descriptions

Field	Description
SADR	Start Address. This shows the word following the Boot ID (with the RCHW field). The boot processor starts executing application code from this address.

5.3.4.10 Code Length Register (SSCM_CLEN)

Offset: 0x002C Access: Read/Write

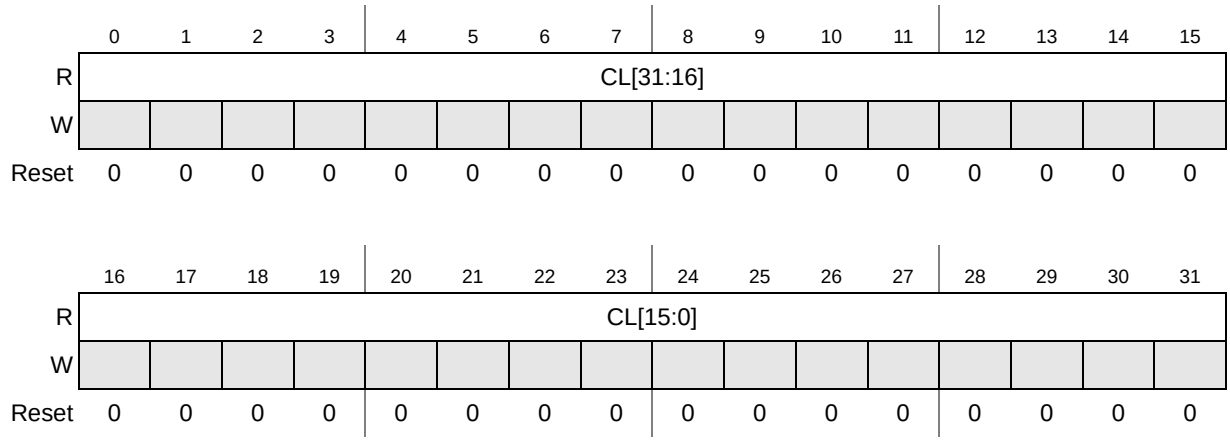


Figure 29. Code Length register (SSCM_CLEN)

Table 37. SSCM_CLEN field descriptions

Field	Description
CL	Length of the code for the identified boot sector. This shows the word following the Boot ID (with the RCHW field) and the application start address in the Flash. For CSE applications, this word must contain the code length. For other applications, this register can be ignored and the word in Flash can be used for other purposes (e.g. application code).

6 Clock Description

This chapter describes the clock architectural implementation for SPC564Bxx and SPC56ECxx.

6.1 Clock architecture

System clocks are generated from three sources:

- Fast external crystal oscillator 4-40 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 30*.

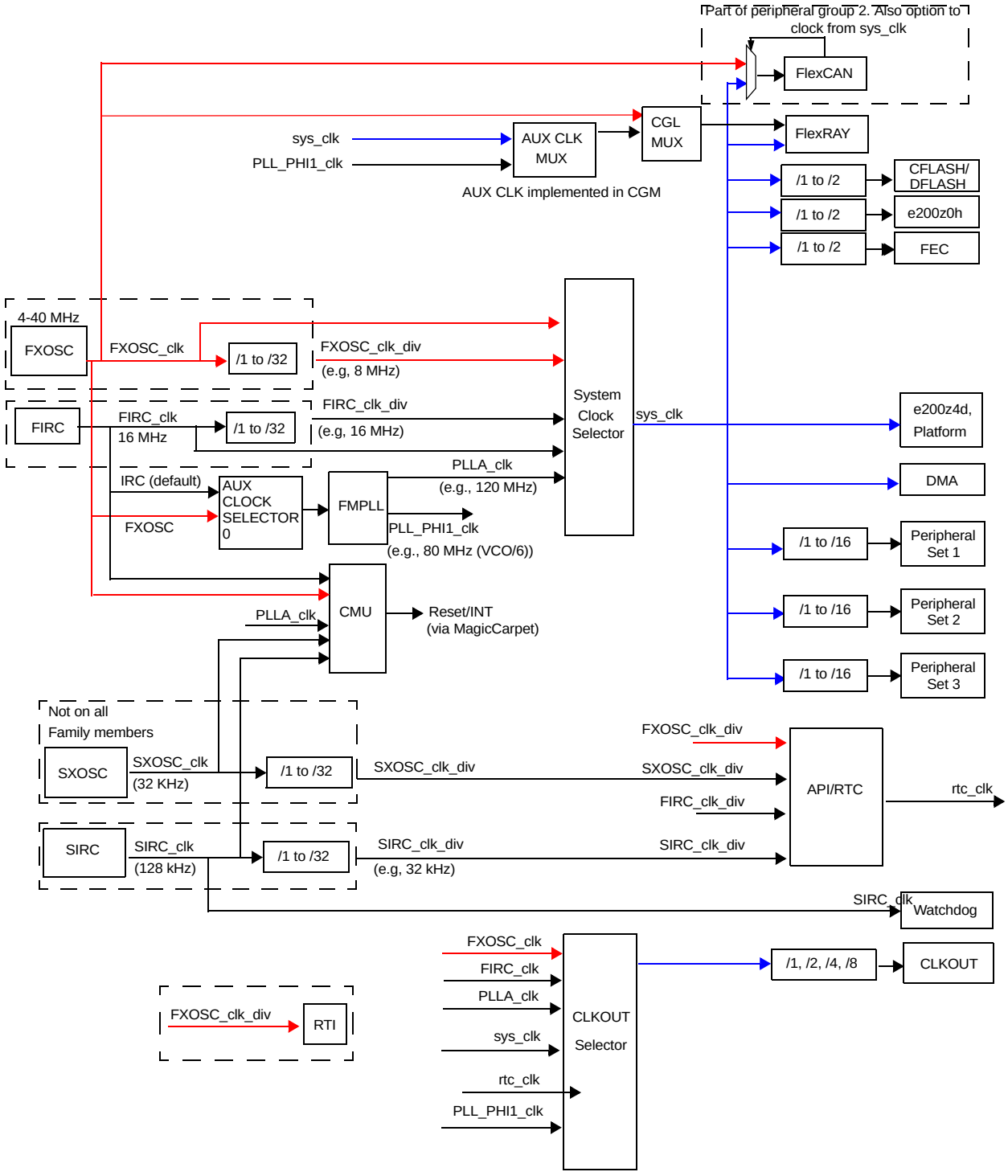


Figure 30. SPC564Bxx and SPC56ECxx system clock generation

1. For clock divider maximum frequency and for example configuration with respect to CFLASH/DFLASH, e200z0h, FEC, and Peripheral SET 1/2/3 blocks in this figure, please refer to *Table 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

6.2 Clock gating

The SPC564Bxx and SPC56ECxx provides the user with the possibility of gating the clock to the peripherals. *Table 38* 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 38. SPC564Bxx and SPC56ECxx — Peripheral clock sources

Peripheral	Register gating address offset (base = 0xC3FDC0C0)(1)	Peripheral set(2)
e200z0h Platform	none (managed through ME mode)	The e200z0h clock can be programmed to be divide by 1 or divide by 2 of the system clock
e200z4d Platform	none (managed through ME mode)	—
DSPI_n	4+n (n = 0..7)	2
FlexCAN_n	16+n (n = 0..5)	2
DMACHMUX	23	—
FlexRay	24	—
ADC_0	32	3
ADC_1	33	3
I^2C	44	1
LINFlexD_n	48+n(n = 0..7)	1
LINFlexD_8	12	1
LINFlexD_9	13	1
CTU	57	3
CANS	60	—
CFLASH_0	66	The flash peripheral clock can be programmed to be divide by 1 or divide by 2 of the system clock
DFLASH	67	The flash peripheral clock can be programmed to be divide by 1 or divide by 2 of the system clock
SIUL	68	—
WKUP	69	—
eMIOS_n	72+n (n = 0..1)	3

Table 38. SPC564Bxx and SPC56ECxx — Peripheral clock sources (continued)

Peripheral	Register gating address offset (base = 0xC3FDC0C0)(1)	Peripheral set(2)
CFLASH_1	76	The flash peripheral clock can be programmed to be divide by 1 or divide by 2 of the system clock
RTC/API	91	—
PIT_RTI	92	—
CMU	104	—

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

Note: In order to alter the e200z0 clock divider, all the potential sources of interrupts to e200z0 should be disabled. They should be disabled at the INTC level as well as WKPU level (by disabling PCTL for WKPU).

Table 39 shows how to setup various dividers for different frequency operations. Dynamic switching of the clock dividers for the peripherals takes effect immediately and affects the external functions.

Table 39. SPC564Bxx and SPC56ECxx example peripheral clock divider setup

System frequency (sys_clk) (MHz)	Peripheral set 1(1) Max freq. = 32 MHz	Peripheral set 2(2) Max freq. = 64 MHz	Peripheral set 3(3) Max freq. = 64 MHz	FEC frequency(4) Max freq. = 80 MHz	e200z0h frequency(5) Max freq. = 80 MHz	Flash memory BIU frequency(6) Max freq. = 80 MHz
120	30	60	60	60	60	60
64	32	64	64	64	64	64
80	20	40	40	80	80	80

1. See the CGM_SC_DC0 section of this reference manual.
2. See the CGM_SC_DC1 section of this reference manual.
3. See the CGM_SC_DC2 section of this reference manual.
4. See the CGM_FEC_DCR section of this reference manual.
5. See the CGM_Z0_DCR section of this reference manual.
6. See the CGM_FLASH_DCR section of this reference manual.

Note: For a system frequency greater than 64 MHz, SRAMC should be programmed for an additional wait state. See RAM WS bit in Section 38.4.2.5: Miscellaneous User-Defined Control Register (MUDCR).

6.3 Fast external crystal oscillator (FXOSC) digital interface

The FXOSC digital interface controls the operation of the 4–40 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 40 shows the truth table of different oscillator configurations.

Table 40. 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

Address: 0xC3FE_0000 Access: Special read/write

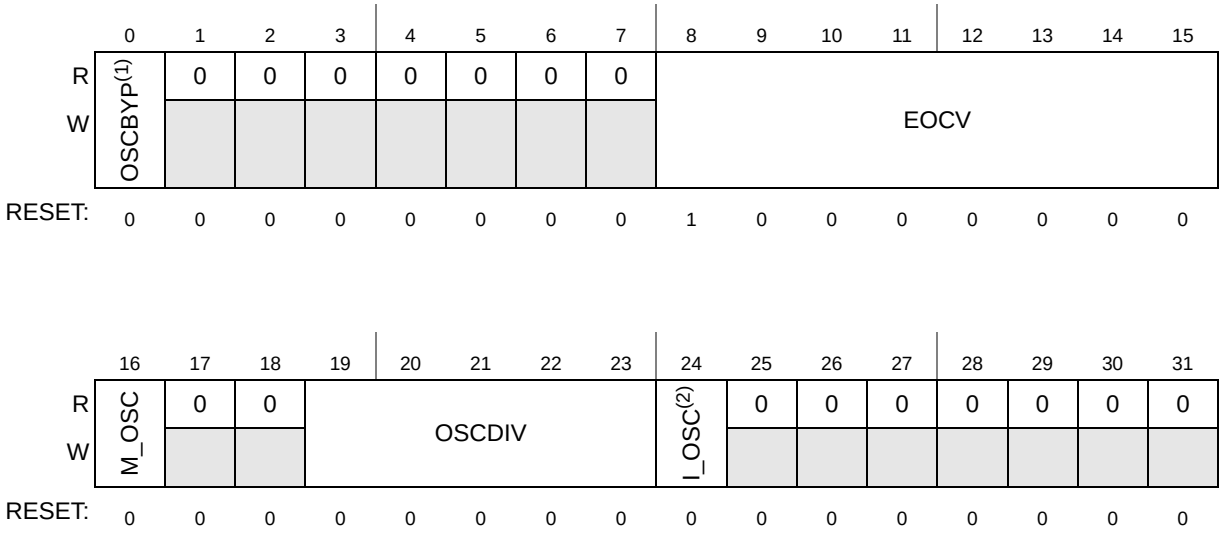


Figure 31. Fast External Crystal Oscillator Control Register (FXOSC_CTL)

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 41. 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.

Note: Bus access errors are generated in only half of the non-implemented address space of Oscillator External Interface (Crystal XOSC) and RCOSC Digital Interface (16MHz Internal RC oscillator [IRC]). Do not access unimplemented address space for XOSC and RCOSC register areas OR write software that is not dependent on receiving an error when access to unimplemented XOSC and RCOSC space occurs

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 clock available interrupt
- 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. Also, an interrupt pending SXOSC_CTL[I_OSC] bit is set. An interrupt will be generated if the interrupt mask bit M_OSC is set.

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 42 shows the truth table of different configurations of the oscillator.

Table 42. 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

Address: 0xC3FE_0040 Access: Special read/write

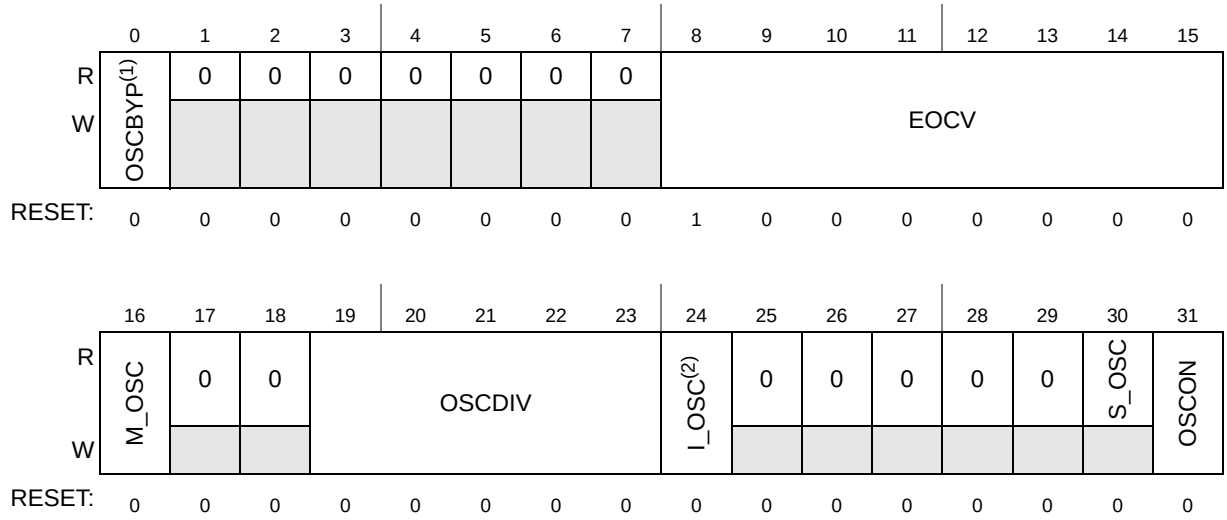


Figure 32. Slow External Crystal Oscillator Control register (SXOSC_CTL)

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 43. 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 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.

Table 43. SXOSC_CTL field descriptions (continued)

Field	Description
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 field is set by hardware when OSCCNT counter reaches the count value EOCV × 512. 0 No oscillator clock interrupt occurred. 1 Oscillator clock interrupt pending.
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 OFF, but can be configured ON 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

Address: 0xC3FE_0080 Access: Read/write

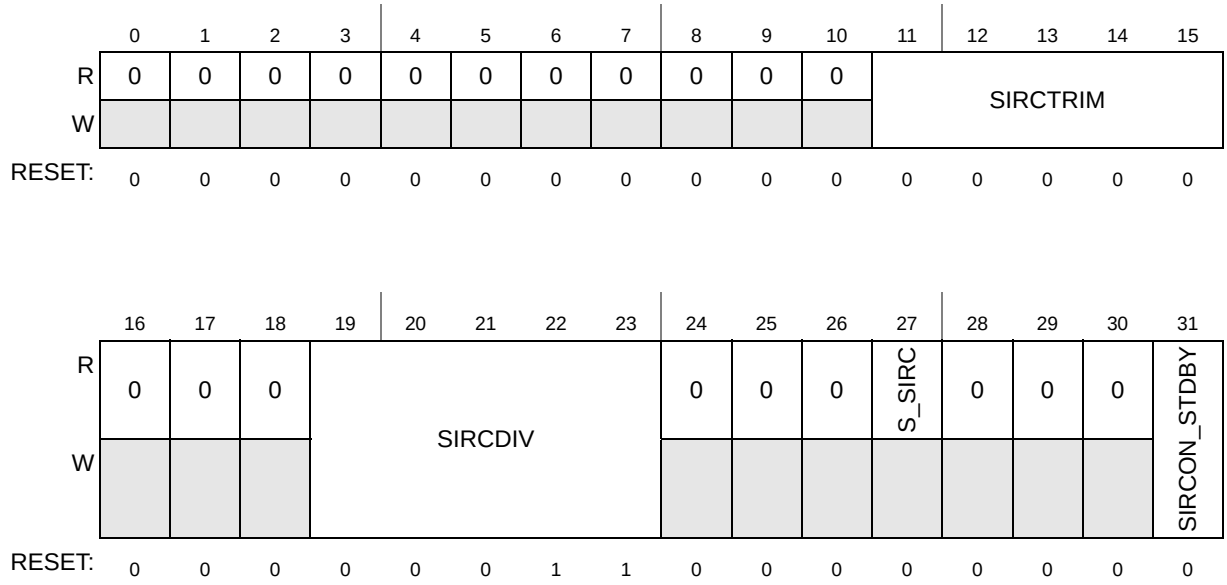


Figure 33. Low Power RC Control Register (SIRC_CTL)

Table 44. 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.
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

Address: 0xC3FE_0060 Access: Read/write

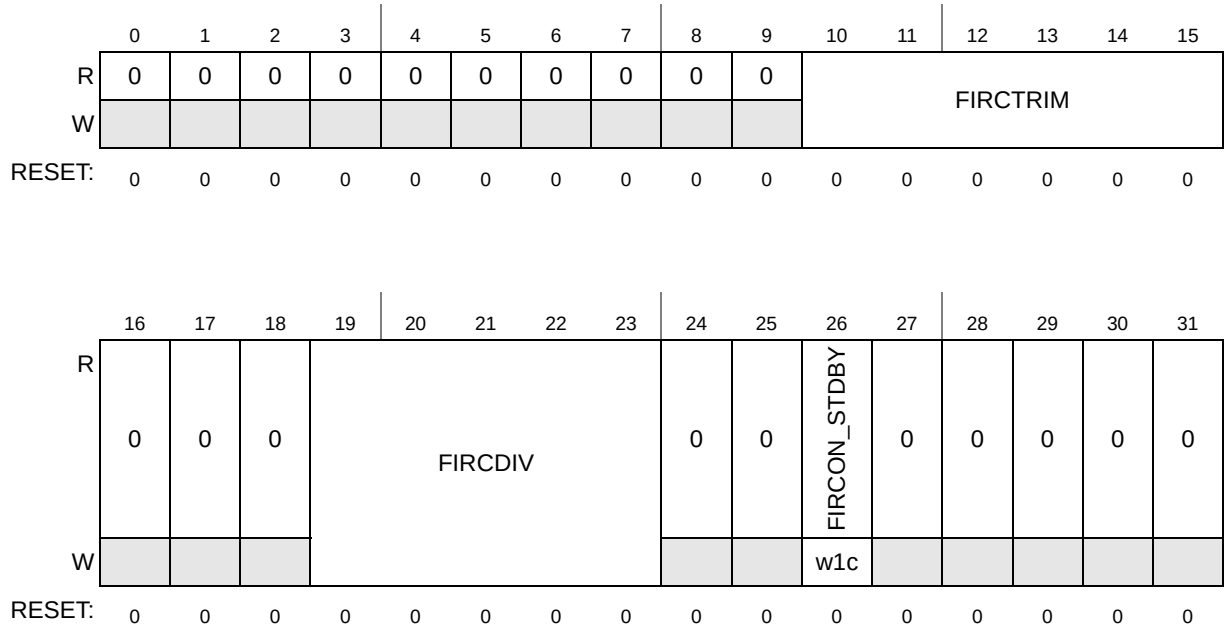


Figure 34. FIRC Oscillator Control Register (FIRC_CTL)

Table 45. 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.
FIRCON_STDBY	FIRC control in STANDBY mode. 0 FIRC is switched off in STANDBY mode. 1 FIRC is in STANDBY mode.

Note: Bus access errors are generated in only half of the non-implemented address space of Oscillator External Interface (Crystal XOSC) and RCOSC Digital Interface (16MHz Internal RC oscillator [IRC]). Do not access unimplemented address space for XOSC and RCOSC register areas OR write software that is not dependent on receiving an error when access to unimplemented XOSC and RCOSC space occurs

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 4–40 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.

SPC564Bxx and SPC56ECxx 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 35*.

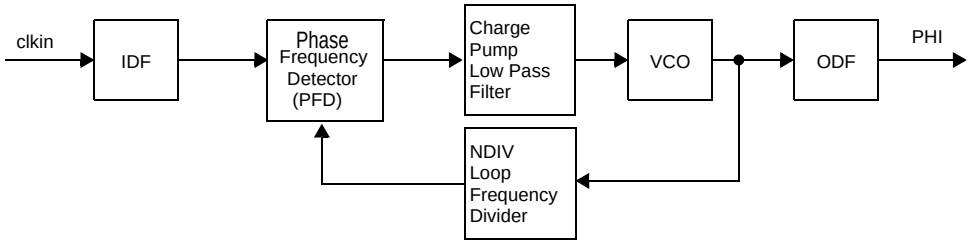


Figure 35. FMPLL block diagram

6.7.3 Features

The FMPLL has the following major features:

- Input clock frequency 4 MHz–40 MHz
- PFD input clock frequency range in normal mode is 4–16 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)

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

  - –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 46 shows the memory map of the FMPLL.

Table 46. FMPLL memory map

Base address: 0xC3FE_00A0		
Address offset	**Register**	**Location**
0x0	*Control Register (CR)*	*on page 186*
0x4	*Modulation Register (MR)*	*on page 188*

6.7.5 Register description

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

f.FMPLL_x are mapped through the ME_CGM register slot

6.7.5.1 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	1	0	0	0	0	0	1

Figure 36. Control Register (CR)

Table 47. CR field descriptions

Field	Description
IDF	The value of this field sets the FMPLL input division factor as described in *Table 48.*
ODF	The value of this field sets the FMPLL output division factor as described in *Table 49.*
NDIV	The value of this field sets the FMPLL loop division factor as described in *Table 50.*
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.
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:** SLOCK=1 indicates that the FMPLL has achieved coarse lock. Fine lock is achieved 200 µs after the FMPLL is enabled.

Table 47. CR field descriptions (continued)

Field	Description
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 48. 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 49. 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 50. 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	—

6.7.5.2 Modulation Register (MR)

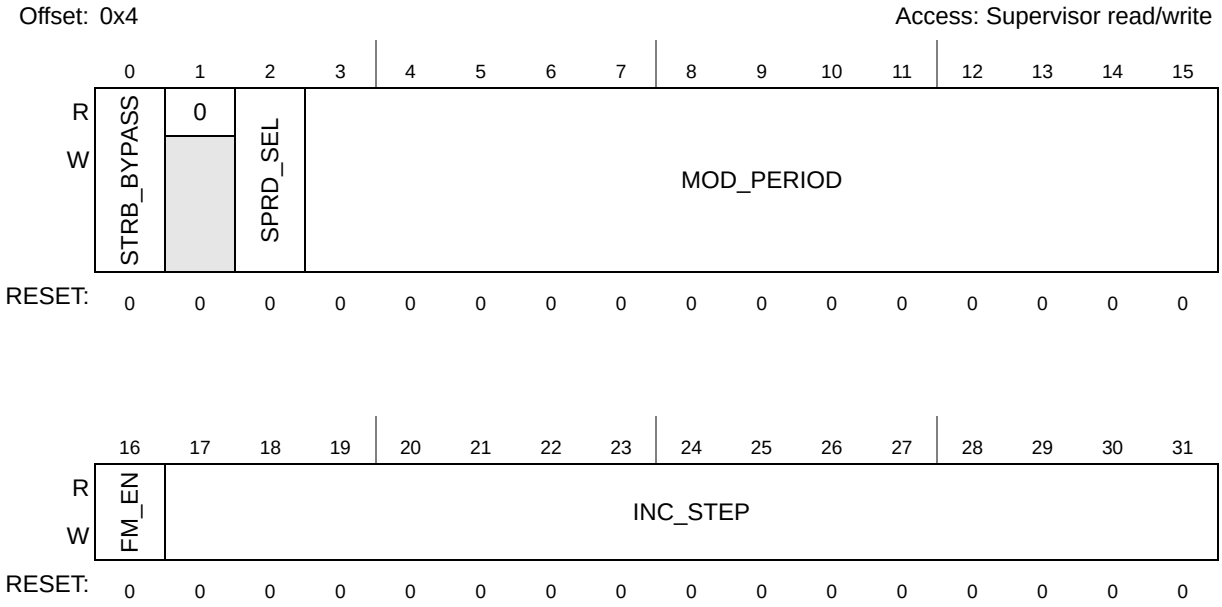


Figure 37. Modulation Register (MR)

Table 51. 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

6.7.6.1 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 (CLKIN) 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 48*, *Table 49* and *Table 50*.

Table 52. FMPLL lookup table

Crystal frequency (MHz)	FMPLL output frequency (MHz)	CR field values			VCO frequency (MHz)	PHI1[1] frequency (MHz)
		IDF	ODF	NDIV		
8	32	0	2	32	256	42.67
	64	0	2	64	512	85.33
	80	0	1	40	320	53.33
	120	0	1	60	480	80
16	32	1	2	32	256	42.67
	64	1	2	64	512	85.33
	80	1	1	40	320	53.33
	120	1	1	60	480	80
40	32	4	2	32	256	42.67
	64	4	2	64	512	85.33
	80	3	1	32	320	53.33
	120	3	1	48	480	80

1. The PHI1 clock is derived from VCO÷6 and can be used to drive FlexRay protocol engine at 80 MHz only (VCO = 480 MHz).

6.7.6.2 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 53*.

Table 53. 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

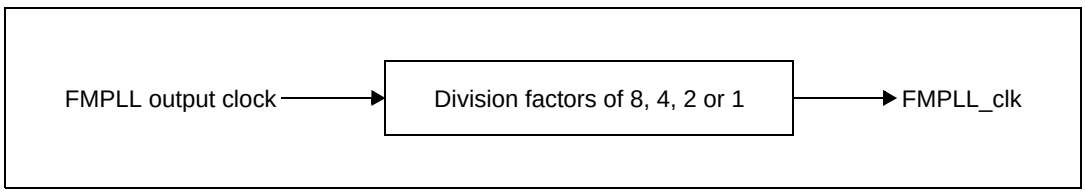


Figure 38. FMPLL output clock division flow during progressive switching

6.7.6.3 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).

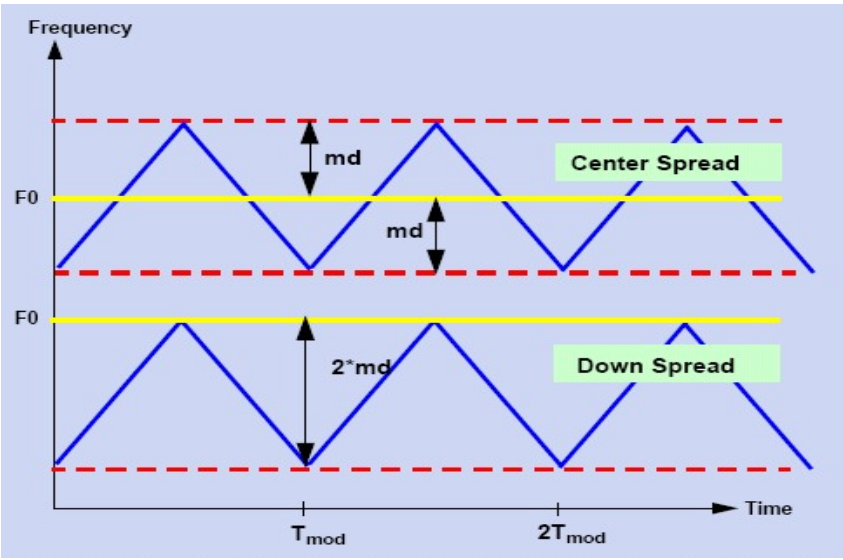


Figure 39. Frequency modulation

6.7.6.4 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.

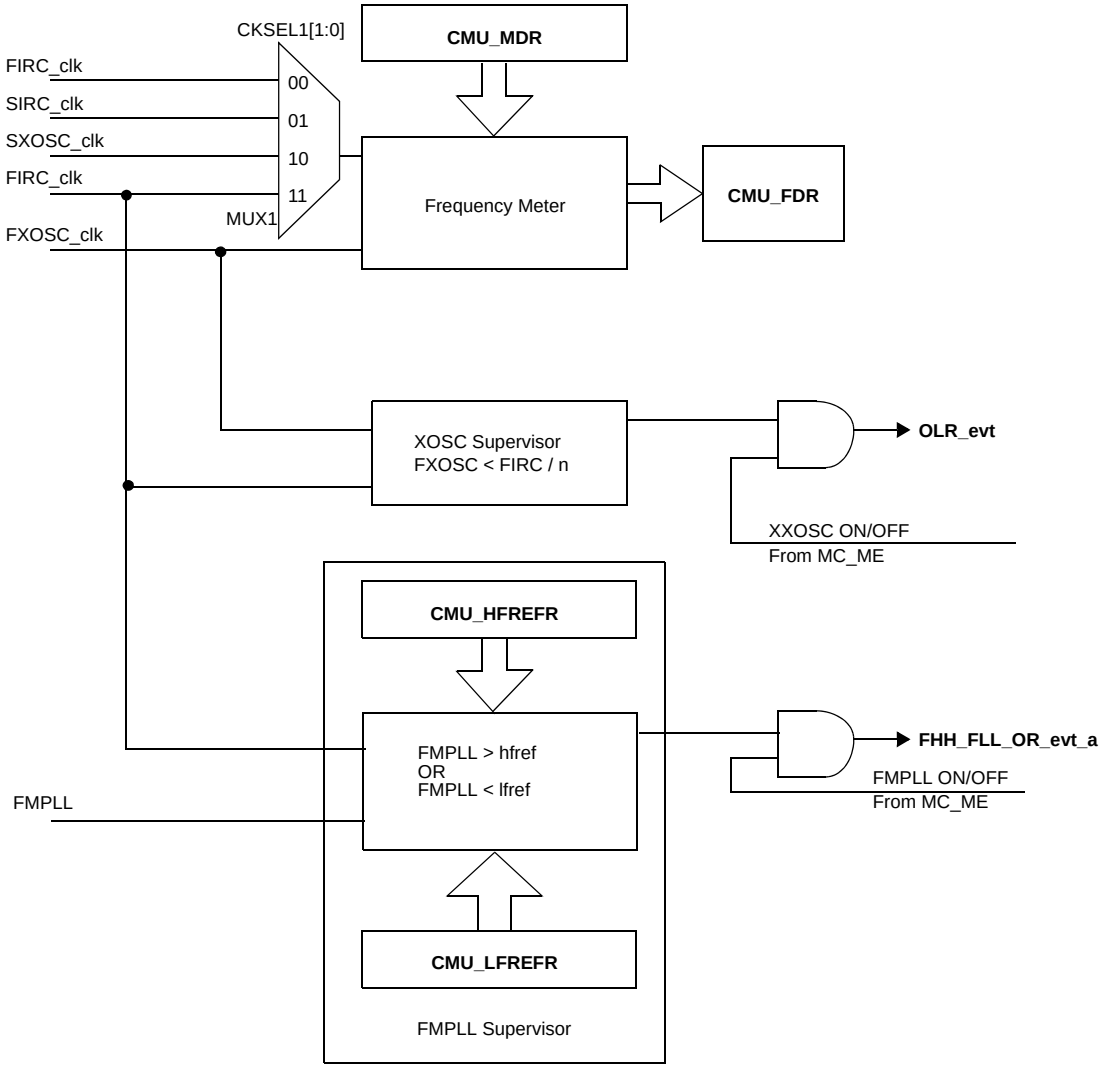
It can also monitor the external crystal oscillator clock, which must be greater than the internal RC clock divided by a division factor given by CMU_CSR[RCDIV], and generates a system clock transition request or an interrupt when enabled.

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 signaling 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 signaling FMPLL failure when asserted. Based on the CMU_HFREFR and CMU_LFREFR configuration, if the FMPLL is greater than high 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 40. 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

6.8.4.1 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 signaled by the MC_ME then:

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

Note: Functional FXOSC monitoring can only be guaranteed when the FXOSC frequency is greater than (FIRC / 2^{RCDIV}) + 0.5 MHz.

Note: The function of the XOSC monitor is not guaranteed when the difference between XOSC and FIRC/2^{RCDIV} (that is, the reference clock frequency) is less than 0.5 MHz.

6.8.4.2 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 signaled by the MC_ME, then:

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

Note: Functional FMPLL monitoring can only be guaranteed when the FMPLL frequency is greater than (FIRC / 4) + 0.5 MHz.

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.

Note: The function of the FMPLL monitor is not guaranteed when the difference between FMPLL and FIRC/4 (that is, the reference clock frequency) is less than 0.5 MHz.

6.8.4.3 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 54*.

Table 54. CMU memory map

Base address: 0xC3FE_0100			
Register name	**Address offset**	**Reset value**	**Location**
Control Status Register (CMU_CSR)	0x00	0x00000006	*on page 196*
Frequency Display Register (CMU_FDR)	0x04	0x00000000	*on page 197*
High Frequency Reference Register FMPLL (CMU_HFREFR)	0x08	0x00000FFF	*on page 198*
Low Frequency Reference Register FMPLL (CMU_LFREFR)	0x0C	0x00000000	*on page 198*
Interrupt Status Register (CMU_ISR)	0x10	0x00000000	*on page 199*
Reserved	0x14	0x00000000	—
Measurement Duration Register (CMU_MDR)	0x18	0x00000000	*on page 199*

g. x = FIRC,SIRC or SXOSC

6.8.5.1 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

Figure 41. Control Status Register (CMU_CSR)

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 55. 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.

6.8.5.2 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

Figure 42. Frequency Display Register (CMU_FDR)

Table 56. 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.

6.8.5.3 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

Figure 43. High Frequency Reference Register FMPLL (CMU_HFREFR)

Table 57. 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 ÷ 16) × (f_{FIRC} ÷ 4).

6.8.5.4 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

Figure 44. Low Frequency Reference Register FMPLL (CMU_LFREFR)

Table 58. 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).

6.8.5.5 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

Figure 45. Interrupt status register (CMU_ISR)

Table 59. 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 signaled by the MC_ME. It can be cleared by software by writing '1'. 0 No FHH event. 1 FHH event is pending.
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 signaled 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 signaled by the MC_ME. It can be cleared by software by writing '1'. 0 No OLR event. 1 OLR event is pending.

6.8.5.6 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

Figure 46. Measurement Duration Register (CMU_MDR)

Table 60. 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.

Note: As per design, MDR=1 is not allowed as the FDR counter counts from 0 to MDR-1.

Note: When FXOSC > FIRC and MDR=0xFFFFF, it will cause overflow of FDR register and the value stored cannot be relied upon.At the IP level, the frequency relationship between the measured and reference frequency in not known and may vary across products.

Note: Based on the frequency relationship in a particular product, application has to decide the appropriate value of MDR to avoid rollover of FDR with decent resolution of measurement.

7 Clock Generation Module (MC_CGM)

7.1 Introduction

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

Figure 47 depicts the MC_CGM block diagram.

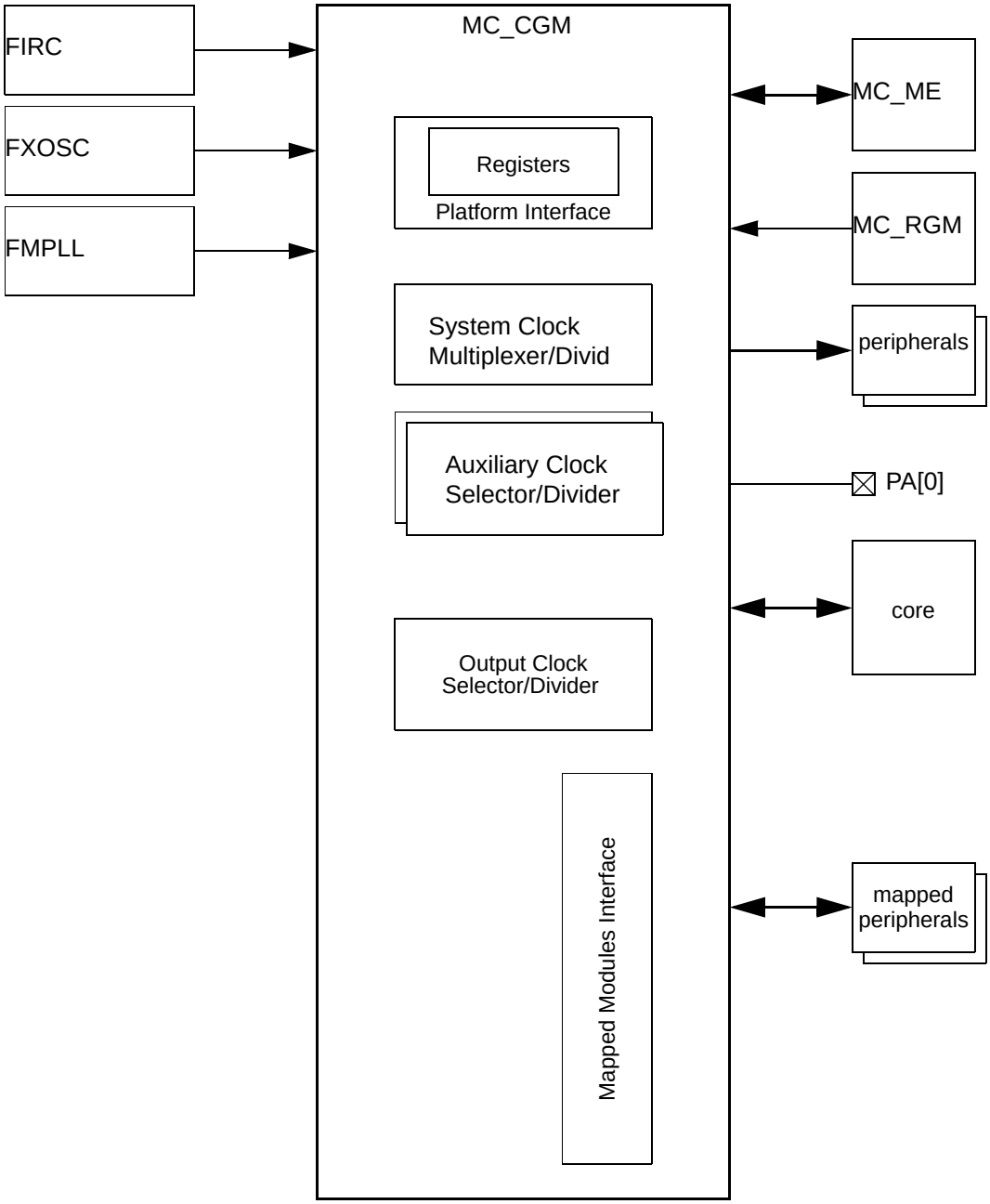


Figure 47. MC_CGM block diagram

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
- contains a set of registers to control peripheral clock selection
- 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 External signal description

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

7.4 Memory Map and Register Definition

Table 61. MC_CGM Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_00C0	CGM_E200Z0H CORE_DCR	e200z0h Core Clock Divider Configuration	byte	read	read/write	read/write	*on page 208*
0xC3FE_00E0	CGM_FEC_DCR	FEC Clock Divider Configuration	byte	read	read/write	read/write	*on page 208*
0xC3FE_0120	CGM_FLASH_DCR	Flash Clock Divider Configuration	byte	read	read/write	read/write	*on page 209*
0xC3FE_0370	CGM_OC_EN	Output Clock Enable	word	read	read/write	read/write	*on page 209*
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read	read/write	read/write	*on page 210*
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	read	read	*on page 211*
0xC3FE_037C	CGM_SC_DC0	System Clock Divider Configuration 0	byte	read	read/write	read/write	*on page 211*
0xC3FE_037D	CGM_SC_DC1	System Clock Divider Configuration 1	byte	read	read/write	read/write	*on page 212*

Table 61. MC_CGM Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_037E	CGM_SC_DC2	System Clock Divider Configuration 2	byte	read	read/write	read/write	*on page 212*
0xC3FE_0380	CGM_AC0_SC	Aux Clock 0 Select Control	word	read	read/write	read/write	*on page 213*
0xC3FE_0388	CGM_AC1_SC	Aux Clock 1 Select Control	word	read	read/write	read/write	*on page 214*
0xC3FE_038C	CGM_AC1_DC0	Aux Clock 1 Divider Configuration 0	byte	read	read/write	read/write	*on page 214*

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

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

Table 62. MC_CGM Memory Map

Address	Name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0000 … 0xC3FE_001C	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																

Table 62. MC_CGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_00A0 … 0xC3FE_00BC	FMPLL registers																	
0xC3FE_00C0	CGM_E200Z0H CORE_DCR	R	0	0	0	0	0	0	0	DIV	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_00C4 … 0xC3FE_00DC	reserved																	
0xC3FE_00E0	CGM_FEC_DCR	R	0	0	0	0	0	0	0	DIV	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_00E4 … 0xC3FE_00FC	reserved																	
0xC3FE_0100 … 0xC3FE_011C	CMU registers																	
0xC3FE_0120	CGM_FLASH_DCR	R	0	0	0	0	0	0	0	DIV	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_0124 … 0xC3FE_013C	reserved																	
0xC3FE_0140 … 0xC3FE_015C	reserved																	

Table 62. MC_CGM Memory Map (continued)

Address	Name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_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																
0xC3FE_0240 … 0xC3FE_025C	reserved																
0xC3FE_0260 … 0xC3FD_C27C	reserved																

Table 62. MC_CGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_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																	
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																

Table 62. MC_CGM Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_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_0380	CGM_AC0_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0384	reserved																	
0xC3FE_0388	CGM_AC1_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_038C	CGM_AC1_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0400 … 0xC3FE_3FFC	reserved																	

7.4.1 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.

7.4.1.1 e200z0h Core Clock Divider Configuration Register (CGM_Z0_DCR)

Address 0xC3FE_00C0 Access: User read, Supervisor read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	DIV
W								
Reset	0	0	0	0	0	0	0	0

Figure 48. e200z0h Core Clock Divider Configuration Register (CGM_Z0_DCR)

This register sets the e200z0h Core clock division factor. For clock divider maximum frequency and for example configuration, please refer to *Table 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

Note: The value of the DIV field of the CGM_Z0_DCR should only be changed when the e200z0h core clock is disabled or the core is in WAIT state. Also, the e200z0h core cannot run at a frequency higher than 80 MHz. If the system clock frequency is greater than 80 MHz, the e200z0h core clock frequency must be the system clock frequency divided by 2 (DIV = 1).Otherwise, the e200z0h core clock frequency can be the same as the system clock frequency (DIV = 0).

Table 63. e200z0h Core Clock Divider Configuration Register (CGM_Z0_DCR) Field Description

Field	Description
DIV	Divider Division Value — The resultant e200z0h Core clock will have a period ‘DIV + 1’ times that of the system clock.

7.4.1.2 FEC Clock Divider Configuration Register (CGM_FEC_DCR)

This register sets the FEC clock division factor. For clock divider maximum frequency and for example configuration, please refer to *TTable 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

Note: For 100 Mbit/s Ethernet performance, the FEC must operate at greater than or equal to 50 MHz. If the system clock frequency is less than or equal to 80 MHz, the FEC clock frequency must be the same as the system clock frequency (DIV = 0). If the system clock frequency is greater than or equal to 100 MHz, the FEC clock frequency must be the system clock divided by 2 (DIV = 1).

Address 0xC3FE_00E0 Access: User read, Supervisor read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	DIV
W								
Reset	0	0	0	0	0	0	0	1

Figure 49. FEC Clock Divider Configuration Register (CGM_FEC_DCR)

Table 64. FEC Clock Divider Configuration Register (CGM_FEC_DCR) Field Description

Field	Description
DIV	Divider Division Value — The resultant FEC clock will have a period ‘DIV + 1’ times that of the system clock.

7.4.1.3 Flash Clock Divider Configuration Register (CGM_FLASH_DCR)

This register sets the Flash clock division factor. For clock divider maximum frequency and for example configuration, please refer to *Table 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

Note: The flash register interface cannot run at a frequency higher than 80 MHz. If the system clock frequency is greater than 80 MHz, the flash clock frequency must be the system clock divided by 2 (DIV = 1). Otherwise, the flash clock frequency can be the same as the system clock frequency (DIV = 0).

Address 0xC3FE_0120 Access: User read, Supervisor read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	DIV
W								
Reset	0	0	0	0	0	0	0	1

Figure 50. Flash Clock Divider Configuration Register (CGM_Flash_DCR)

Table 65. Flash Clock Divider Configuration Register (CGM_Flash_DCR) Field Description

Field	Description
DIV	Divider Division Value — The resultant Flash clock will have a period ‘DIV + 1’ times that of the system clock.

7.4.1.4 Output Clock Enable Register (CGM_OC_EN)

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

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

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

Figure 51. Output Clock Enable Register (CGM_OC_EN)

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

Table 66. 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

7.4.1.5 Output Clock Division Select Register (CGM_OCDS_SC)

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

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

Figure 52. Output Clock Division Select Register (CGM_OCDS_SC)

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 67. 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 FXOSC 0001 FIRC 0010 FMPLL 0011 system clock 0100 RTC clock 0101 FMPLL_PHI1 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

7.4.1.6 System Clock Select Status Register (CGM_SC_SS)

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

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

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

Figure 53. System Clock Select Status Register (CGM_SC_SS)

This register provides the current system clock source selection.

Table 68. 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 FIRC 0001 FIRC_divided 0010 FXOSC 0011 FXOSC_divided 0100 FMPLL 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

7.4.1.7 System Clock Divider 0 Configuration Register (CGM_SC_DC0)

Address 0xC3FE_037C Access: User read, Supervisor read/write, Test read/write

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

Figure 54. System Clock Divider 0 Configuration Register (CGM_SC_DC0)

This register controls system clock divider 0. For clock divider maximum frequency and for example configuration, please refer to *Table 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

Table 69. System Clock Divider 0 Configuration Register (CGM_SC_DC0) 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.

7.4.1.8 System Clock Divider 1 Configuration Register (CGM_SC_DC1)

Address 0xC3FE_037D Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE1	0	0	0	DIV1			
W								
Reset	1	0	0	0	0	0	0	0

Figure 55. System Clock Divider 1 Configuration Register (CGM_SC_DC1)

This register controls system clock divider 1. This register controls system clock divider 0. For clock divider maximum frequency and for example configuration, please refer to *Table 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

Table 70. System Clock Divider 1 Configuration Register (CGM_SC_DC1) Field Descriptions

Field	Description
DE1	Divider 1 Enable. 0 Disable system clock divider 1 1 Enable system clock divider 1
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.

7.4.1.9 System Clock Divider 2 Configuration Register (CGM_SC_DC2)

Address 0xC3FE_037E Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE2	0	0	0	DIV2			
W								
Reset	1	0	0	0	0	0	0	0

Figure 56. System Clock Divider 2 Configuration Register (CGM_SC_DC2)

This register controls system clock divider 2. For clock divider maximum frequency and for example configuration, please refer to *Table 39: SPC564Bxx and SPC56ECxx example peripheral clock divider setup*.

Table 71. System Clock Divider 2 Configuration Register (CGM_SC_DC2) Field Descriptions

Field	Description
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.

7.4.1.10 Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

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

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

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

Figure 57. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

This register is used to select the current clock source for the FMPLL reference clock.

See *Figure 61* for details.

Table 72. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 0 Source Selection Control** — This value selects the current source for auxiliary clock 0. 0000 FXOSC 0001 FIRC 0010 reserved 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

7.4.1.11 Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

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

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

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

Figure 58. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

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

- divided by auxiliary clock 1 divider 0: FlexRay clock

Table 73. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) Field Descriptions

Field	Description
SELCTL	**Auxiliary Clock 1 Source Selection Control** — This value selects the current source for auxiliary clock 1. 0000 system clock 0001 FMPLL_PHI1 0010 reserved 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved **Note:** Before changing the clock source, the FlexRay module should be disabled to prevent any communication glitches.

7.4.1.12 Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0)

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

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

Figure 59. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0)

This register controls auxiliary clock 1 divider 0.

Table 74. Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0) Field Descriptions

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

7.5 Functional description

7.5.1 System clock generation

Figure 60 shows the block diagram of the system clock generation logic. The MC_ME provides the system clock select and switch mask (see MC_ME documentation for more details), and the MC_RGM provides the safe clock request (see MC_RGM documentation for more details). The safe clock request forces the selector to select the FIRC as the system clock and to ignore the system clock select.

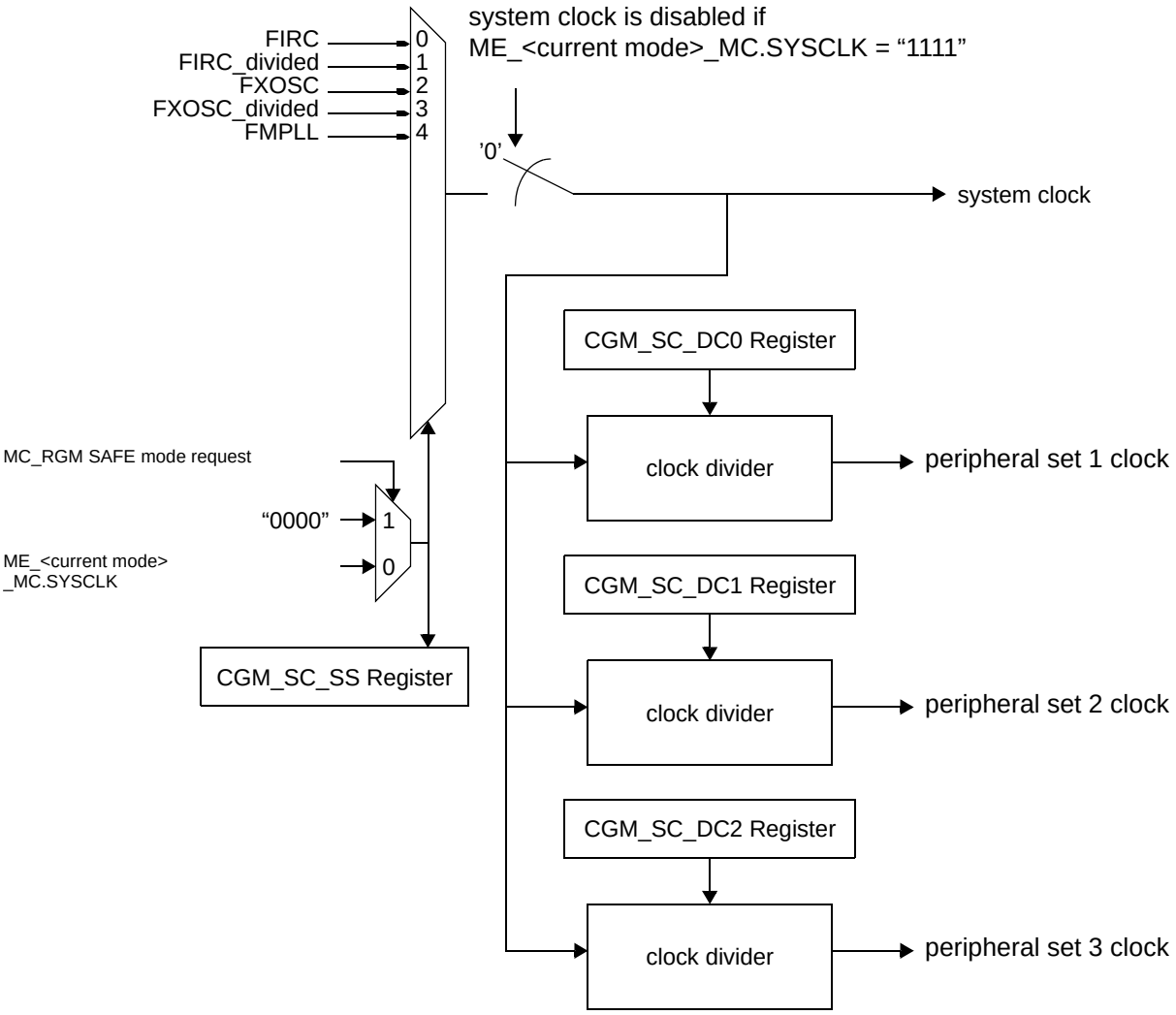


Figure 60. MC_CGM System Clock Generation Overview

7.5.1.1 System clock source selection

During normal operation, the system clock selection is controlled

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

7.5.1.2 System clock disable

During the STOP0 and TEST modes normal, the system clock can be disabled by the MC_ME.

7.5.1.3 System clock dividers

The MC_CGM generates the following derived clocks from the system clock:

- peripheral set 1 clock - controlled by the CGM_SC_DC0 register
- peripheral set 2 clock - controlled by the CGM_SC_DC1 register
- peripheral set 3 clock - controlled by the CGM_SC_DC2 register

7.5.2 Auxiliary clock generation

Figure 60 shows the block diagram of the auxiliary clock generation logic. See *Section 7.4.1.10: Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)*and *Section 7.4.1.11: Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)* for auxiliary clock selection control.

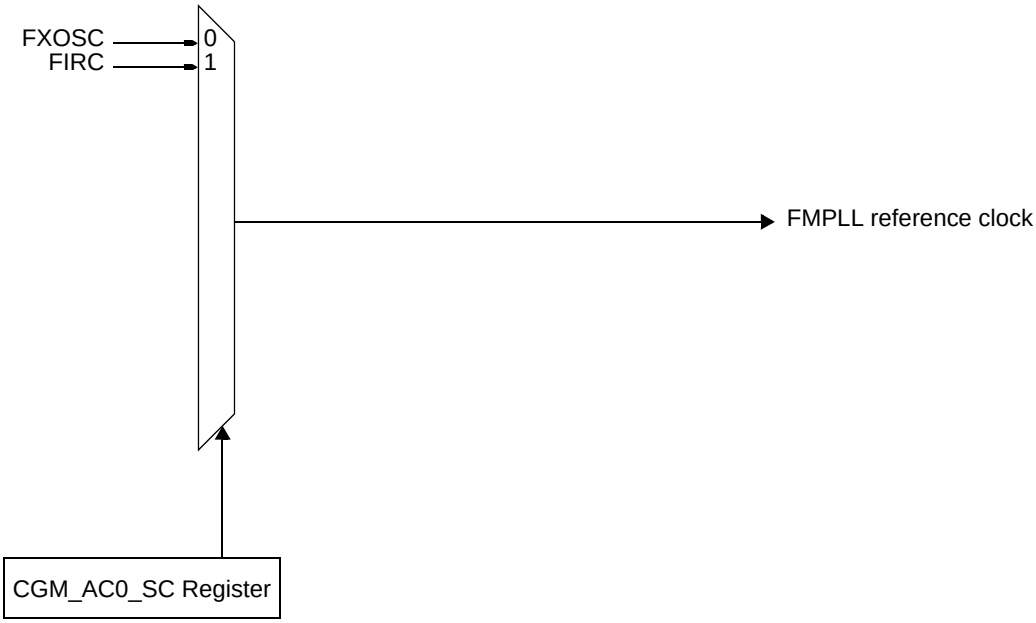


Figure 61. MC_CGM Auxiliary Clock 0 Generation Overview

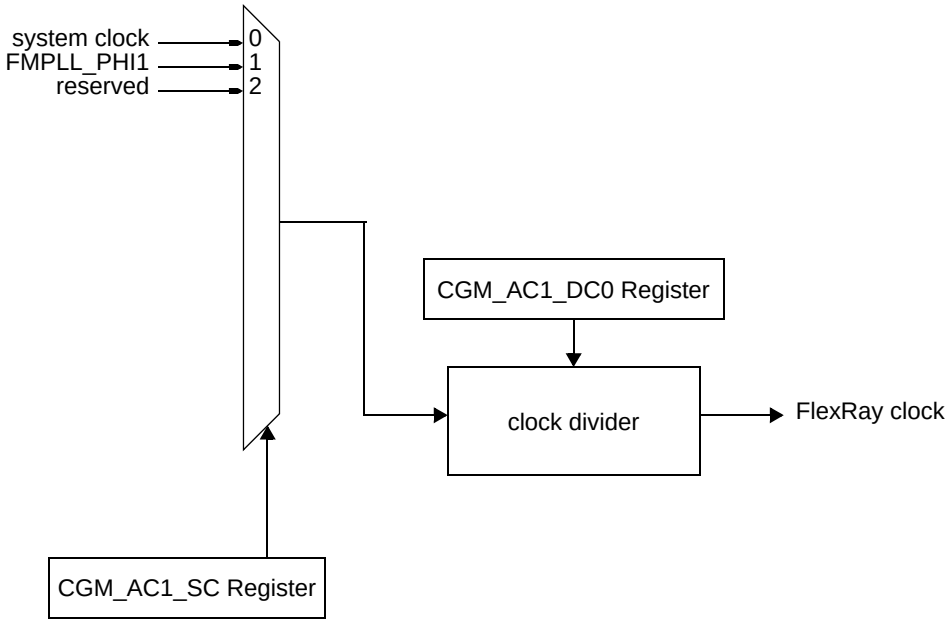


Figure 62. MC_CGM Auxiliary Clock 1 Generation Overview

7.5.2.1 Auxiliary Clock Source Selection

During normal operation, the auxiliary clock selection is done via the CGM_AC0…1_SC registers. If software selects an 'unavailable' source, the old selection remains, and the register content does not change.

7.5.2.2 Auxiliary Clock Dividers

- The MC_CGM generate the following derived clocks:

FlexRay clock - controlled by the CGM_AC1_DC0 register

Note: When Flexray is clocked through FMPLL.PHI1, then it should be ensured that the Flexray clock is gated using CGM_AC1_DC0[DIV0] register before entering into STOP mode.

7.5.3 Dividers functional description

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

- *Section 7.4.1.7: System Clock Divider 0 Configuration Register (CGM_SC_DC0)*
- *Section 7.4.1.8: System Clock Divider 1 Configuration Register (CGM_SC_DC1)*
- *Section 7.4.1.9: System Clock Divider 2 Configuration Register (CGM_SC_DC2)*
- *Section 7.4.1.12: Auxiliary Clock 1 Divider 0 Configuration Register (CGM_AC1_DC0)*

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

7.5.4 Output Clock Multiplexing

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

7.5.5 Output Clock Division Selection

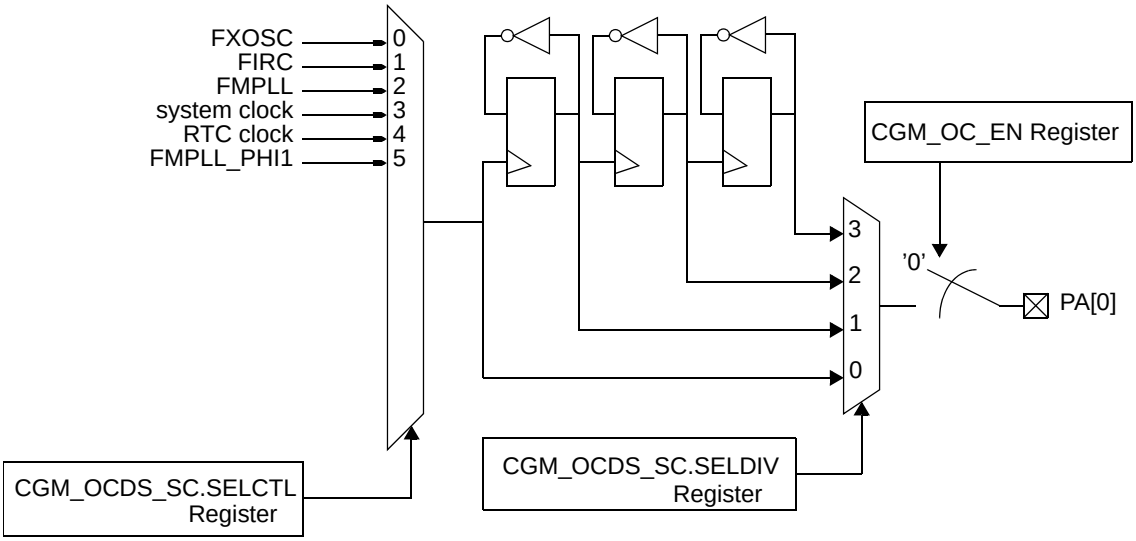


Figure 63. MC_CGM Output Clock Multiplexer and PA[0] Generation

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

- PA[0] (see *Figure 63*). This signal is generated by using one of the 3-stage ripple counter outputs or the selected signal without division. The non-divided signal is not guaranteed to be 50% duty cycle by the MC_CGM.

the MC_CGM also has an output clock enable register (see *Section 7.4.1.4: 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 microcontroller mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application.

The microcontroller must have one core that is in control of the mode transitions. This would typically be the e200z4 core. It is the responsibility of this core to keep track of:

- The state of the other core
- The operating mode requirements for the other core
- The peripheral configuration requirements for the other core

The e200z4 core needs to configure and enter modes based on these requirements. It must also ensure that a mode change does not prevent the other core from being able to complete its current tasks due to, for example, a peripheral being turned off.

Figure 64 depicts the MC_ME block diagram.

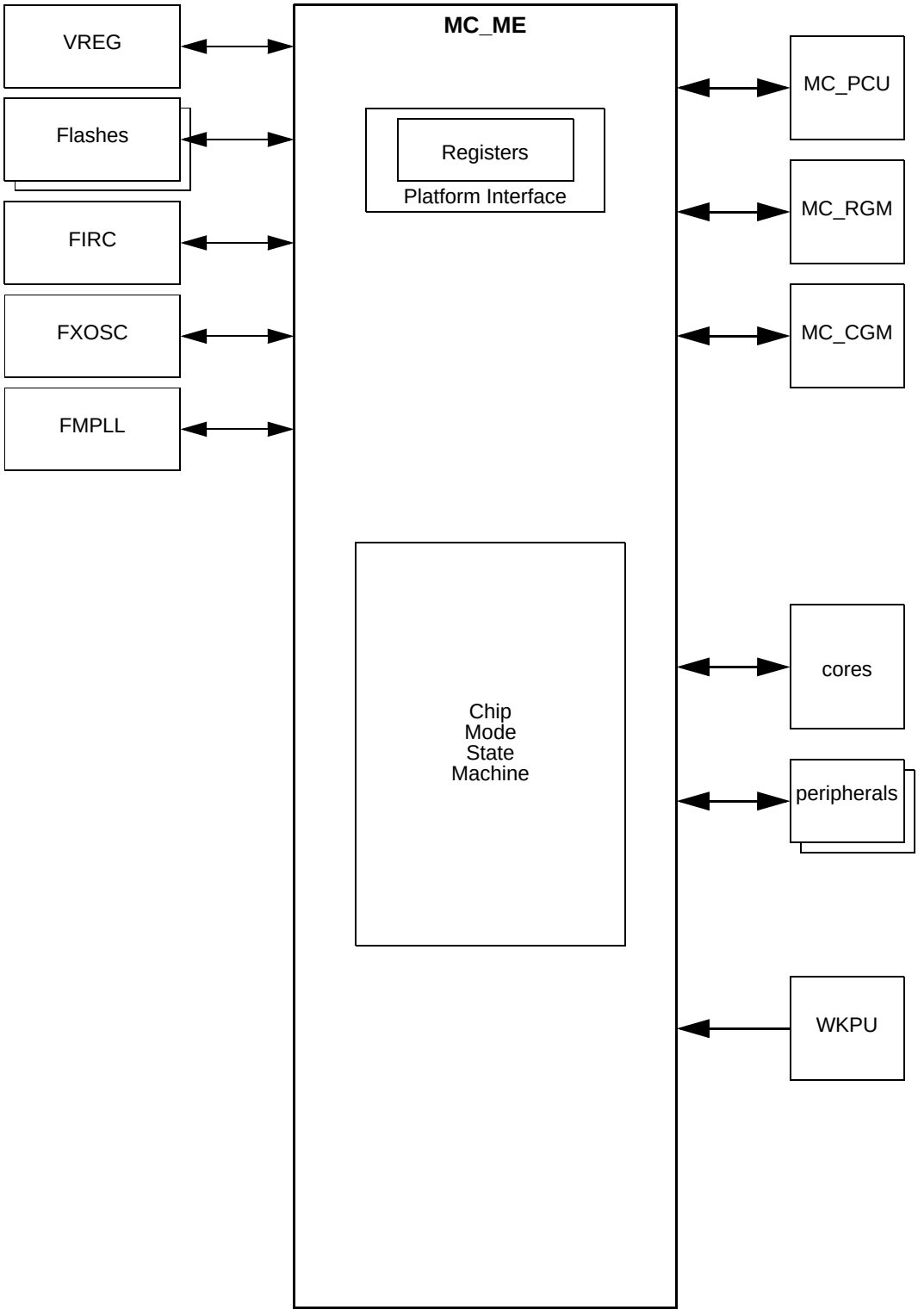


Figure 64. MC_ME block diagram

8.1.2 Features

The MC_ME includes the following features:

- Control of the available modes by the mode enable (ME_ME) register
- Change of current mode by the mode control (ME_MCTL) register
- Mode configuration for each of the available modes by the ME_<mode>_MC registers
- Visibility of the current operating mode and mode configuration by the global status (ME_GS) register
- Configuration of optional interrupts related to mode and mode transition via the interrupt status (ME_IS) and interrupt mask (ME_IM) registers
- Peripheral clock gating control for run and low power modes via the run peripheral configuration (ME_RUN_PC[0..7]), low power peripheral configuration (ME_LP_PC[0..7]) and the peripheral control (ME_PCTLn) registers
- Status Information on the operational status of the peripherals via the peripheral status registers (ME_PS[0..3])

8.1.3 Modes of operation

The MC_ME provides different low power and operational modes which can be configured based on how you want to use the microcontroller. 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:

- System modes are modes that can generally not be configured (with the exception of DRUN mode) and are present to facilitate the startup, error recovery and monitoring of the system:
  - RESET
  - DRUN (default RUN; this mode can be configured by the user)
  - SAFE
  - TEST
- User modes are modes that have to be entered via software and can be highly configured to the applications performance and power requirements:
  - RUN[0..3]
  - HALT
  - STOP
  - STANDBY

Table 75 describes the MC_ME modes. See *Figure 88* for a flowchart of possible mode transitions.

Table 75. MC_ME mode descriptions

Name	Description	Entry	Exit
RESET	This is a mode during which the application is not active. The system remains in this mode until all resources are available for the embedded software to take control of the chip. It manages hardware initialization of chip configuration, voltage regulators, clock sources, and flash modules.	system reset assertion from MC_RGM	system reset deassertion from MC_RGM
DRUN	This is the entry mode for the embedded software. It provides full accessibility to the system and enables the configuration of the system at startup. It provides the unique gate to enter user modes. If the BAM code is executed during boot, this is done so 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 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
TEST	This is a service mode which is intended to provide a control environment for chip software testing.	software request from DRUN	system reset assertion, DRUN via software
RUN0...3	These are software running modes where most processing activity is done. The RUN modes are where most of the processing activity is done. Low power modes can only be entered via one of the RUN modes. The configuration options available to each of the 4 RUN modes are identical. Different clock and power configurations can be configured for each RUN mode to allow a totally configurable RUN environment.	software request from DRUN or other RUN0…3, 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 clock sources, flash, main regulator, etc. for efficient power management at the cost of higher wakeup latency.	software request from RUN0…3	system reset assertion, SAFE on hardware failure, RUN0…3 on interrupt event
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 clock sources for efficient power management at the cost of higher wakeup latency.	software request from RUN0…3	system reset assertion, SAFE on hardware failure, RUN0…3 on interrupt event or wakeup event[1]
STANDBY	STANDBY mode is the lowest power consumption mode. Most of the power to the microcontroller is removed leaving only necessary modules enabled that are required for wakeup. 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

SAFE on failure which was configured in MC_RGM_FEAR to generate safe mode request. Note: This is possible only if MC_STOP0_MC[SYSCLK] was not configured for F.

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

8.3.1 Memory map

Table 76. MC_ME register description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C000	ME_GS	Global Status	word	read	read	read	*on page 233*
0xC3FD_C004	ME_MCTL	Mode Control	word	read	read/write	read/write	*on page 235*
0xC3FD_C008	ME_ME	Mode Enable	word	read	read/write	read/write	*on page 236*
0xC3FD_C00C	ME_IS	Interrupt Status	word	read	read/write	read/write	*on page 238*
0xC3FD_C010	ME_IM	Interrupt Mask	word	read	read/write	read/write	*on page 240*
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read	read/write	read/write	*on page 241*
0xC3FD_C018	ME_DMTS	Debug Mode Transition Status	word	read	read	read	*on page 242*
0xC3FD_C020	ME_RESET_MC	RESET Mode Configuration	word	read	read	read	*on page 244*
0xC3FD_C024	ME_TEST_MC	TEST Mode Configuration	word	read	read/write	read/write	*on page 245*
0xC3FD_C028	ME_SAFE_MC	SAFE Mode Configuration	word	read	read/write	read/write	*on page 246*
0xC3FD_C02C	ME_DRUN_MC	DRUN Mode Configuration	word	read	read/write	read/write	*on page 246*
0xC3FD_C030	ME_RUN0_MC	RUN0 Mode Configuration	word	read	read/write	read/write	*on page 247*
0xC3FD_C034	ME_RUN1_MC	RUN1 Mode Configuration	word	read	read/write	read/write	*on page 247*

Table 76. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C038	ME_RUN2_MC	RUN2 Mode Configuration	word	read	read/write	read/write	on page 247
0xC3FD_C03C	ME_RUN3_MC	RUN3 Mode Configuration	word	read	read/write	read/write	on page 247
0xC3FD_C040	ME_HALT_MC	HALT Mode Configuration	word	read	read/write	read/write	on page 248
0xC3FD_C048	ME_STOP_MC	STOP Mode Configuration	word	read	read/write	read/write	on page 248
0xC3FD_C054	ME_STANDBY_MC	STANDBY Mode Configuration	word	read	read/write	read/write	on page 249
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	read	read	on page 252
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	read	read	on page 253
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	read	read	on page 253
0xC3FD_C06C	ME_PS3	Peripheral Status 3	word	read	read	read	on page 254
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read	read/write	read/write	on page 254
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read	read/write	read/write	on page 254
…							
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read	read/write	read/write	on page 254
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read	read/write	read/write	on page 255
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read	read/write	read/write	on page 255
…							
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read	read/write	read/write	on page 255
0xC3FD_C0C4	ME_PCTL4	DSPI0 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0C5	ME_PCTL5	DSPI1 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0C6	ME_PCTL6	DSPI2 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0C7	ME_PCTL7	DSPI3 Control	byte	read	read/write	read/write	on page 256

Table 76. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C0C8	ME_PCTL8	DSPI4 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0C9	ME_PCTL9	DSPI5 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0CA	ME_PCTL10	DSPI6 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0CB	ME_PCTL11	DSPI7 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0CC	ME_PCTL12	LINFlexD_8 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0CD	ME_PCTL13	LINFlexD_9 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D0	ME_PCTL16	FlexCAN0 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D1	ME_PCTL17	FlexCAN1 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D2	ME_PCTL18	FlexCAN2 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D3	ME_PCTL19	FlexCAN3 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D4	ME_PCTL20	FlexCAN4 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D5	ME_PCTL21	FlexCAN5 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D7	ME_PCTL23	DMA_CH_MUX Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0D8	ME_PCTL24	FlexRay Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0E0	ME_PCTL32	ADC0 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0E1	ME_PCTL33	ADC1 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0EC	ME_PCTL44	I2C_DMA0 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F0	ME_PCTL48	LINFlexD_0 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F1	ME_PCTL49	LINFlexD_1 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F2	ME_PCTL50	LINFlexD_2 Control	byte	read	read/write	read/write	on page 256

Table 76. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C0F3	ME_PCTL51	LINFlexD_3 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F4	ME_PCTL52	LINFlexD_4 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F5	ME_PCTL53	LINFlexD_5 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F6	ME_PCTL54	LINFlexD_6 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F7	ME_PCTL55	LINFlexD_7 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0F9	ME_PCTL57	CTUL Control	byte	read	read/write	read/write	on page 256
0xC3FD_C0FC	ME_PCTL60	CANSampler Control	byte	read	read/write	read/write	on page 256
0xC3FD_C104	ME_PCTL68	SIUL Control	byte	read	read/write	read/write	on page 256
0xC3FD_C105	ME_PCTL69	WKPU Control	byte	read	read/write	read/write	on page 256
0xC3FD_C108	ME_PCTL72	eMIOS0 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C109	ME_PCTL73	eMIOS1 Control	byte	read	read/write	read/write	on page 256
0xC3FD_C11B	ME_PCTL91	RTC/API Control	byte	read	read/write	read/write	on page 256
0xC3FD_C11C	ME_PCTL92	PIT_RTI Control	byte	read	read/write	read/write	on page 256
0xC3FD_C128	ME_PCTL104	CMU Control	byte	read	read/write	read/write	on page 256

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 77. MC_ME memory map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	0	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
		W																
		R	0	0	0	0	0	0	0	0	0	S_FMPLL	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	RESET_DEST	0	STANDBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET_FUNC
		W																
0xC3FD_C00C	ME_IS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	I_ICONF_CU	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C010	ME_IM	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	M_ICONF_CU	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																

Table 77. MC_ME memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C014	ME_IMTS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C018	ME_DMTS	R	PREVIOUS_MODE				0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
		W																
		R	0	VREG_CSRC_SC	CSRC_CSRC_SC	FIRC_SC	SCSRC_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	0	0	0	0	0	0	0	0	0	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																

Table 77. MC_ME memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	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																
		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																

Table 77. MC_ME memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_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	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C058 … 0xC3FD_C05C	reserved																	
0xC3FD_C060	ME_PS0	R	0	0	0	0	0	0	0	S_FlexRay	S_DMA_CH_MUX		S_FlexCAN5	S_FlexCAN4	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
		W																
		R	0	0	S_LINFlexD_9	S_LINFlexD_8	S_DSPI7	S_DSPI6	S_DSPI5	S_DSPI4	S_DSPI3	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	0	0	S_CANSampler	0	0	S_CTUL	0	S_LINFlexD_7	S_LINFlexD_6	S_LINFlexD_5	S_LINFlexD_4	S_LINFlexD_3	S_LINFlexD_2	S_LINFlexD_1	S_LINFlexD_0
		W																
		R	0	0	0	S_I2C	0	0	0	0	0	0	0	0	0	0	S_ADC1	S_ADC0
		W																

Table 77. MC_ME memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT_RTI	S_RTC/API	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	S_eMIOS1	S_eMIOS0	0	0	S_WKPU	S_SIUL	0	0	0	0
		W																
0xC3FD_C06C	ME_PS3	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	S_CMU	0	0	0	0	0	0	0	0
		W																
0xC3FD_C070	reserved																	
0xC3FD_C074 … 0xC3FD_C07C	reserved																	
0xC3FD_C080 … 0xC3FD_C09C	ME_RUN_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		W																
0xC3FD_C0A0 … 0xC3FD_C0BC	ME_LP_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	STANDBY	0	0	STOP	0	HALT	0	0	0	0	0	0	0	0
		W																

Table 77. MC_ME memory map (continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	4 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FD_C0C0 … 0xC3FD_C14C	ME_PCTL0… 143(1)	R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG		RUN_CFG			
		W																
		R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG		RUN_CFG			
		W																
0xC3FD_C150 … 0xC3FD_FFFC	reserved																	

1. There is space in the register map for 144 peripherals. Please refer to *Table 76* for the ME_PCTLn locations actually occupied. The unoccupied locations contain a read-only byte value of 0x00.

8.3.2 Register description

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

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

8.3.2.1 Global Status Register (ME_GS)

This register contains global mode status.

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

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

Figure 65. Global Status Register (ME_GS)

Table 78. Global Status Register (ME_GS) Field Descriptions

Field	Description
S_CURRENT_MODE	Current chip mode status 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 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_PDO	Output power-down status — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and the pads power sequence driver is disabled. The inputs are level unchanged. In STOP mode, only the pad power sequence driver is disabled, but the state of the output remains functional. 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 Reserved 01 Data flash is in power-down mode 10 Reserved 11 Data flash is in normal mode and available for use
S_CFLA	Code flash availability status 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 and available for use

Table 78. Global Status Register (ME_GS) Field Descriptions (continued)

Field	Description
S_FMPLL	FMPLL status 0 FMPLL is not stable 1 FMPLL is providing a stable clock **Note:** S_FMPLL indicates that the FMPLL has achieved coarse lock. Fine lock is achieved 200 µs after the FMPLL is powered on.
S_FXOSC	FXOSC (4-40 MHz external oscillator) status 0 FXOSC (4-40 MHz external oscillator) is not stable 1 FXOSC (4-40 MHz external oscillator) is providing a stable clock
S_FIRC	FIRC status 0 FIRC is not stable 1 FIRC is providing a stable clock
S_SYSCLK	System clock switch status — These bits specify the system clock currently used by the system. 0000 FIRC 0001 FIRC_divided 0010 FXOSC 0011 FXOSC_divided 0100 FMPLL 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

8.3.2.2 Mode Control Register (ME_MCTL)

This register is used to trigger software-controlled mode changes. In order to change mode, the ME_MCTL register must be written with the correct value in the KEY field and then again with the correct "inverted key" value. If an attempt is made to change to a mode that is not enabled in the ME_ME register, then the mode change will not be successful. If "invalid mode" interrupts are enabled, then this will result in an interrupt being raised.

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

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

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

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

Figure 66. Mode Control Register (ME_MCTL)

Table 79. Mode Control Register (ME_MCTL) Field Descriptions

Field	Description
TARGET_MODE	T**arget chip mode** — These bits provide the target chip mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. These bits are automatically updated by hardware while entering SAFE on hardware request. Also, while exiting from the HALT and STOP modes on hardware exit events, these are updated with the appropriate RUN0…3 mode value. 0000 RESET (triggers a 'functional' reset event) 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT 1001 reserved 1010 STOP 1011 reserved 1100 reserved 1101 STANDBY 1110 reserved 1111 RESET (triggers a 'destructive' reset event)
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)

8.3.2.3 Mode Enable Register (ME_ME)

This register allows the user modes to be enabled (with the exception of RUN0, which is always enabled) so that they can be used. Any attempt to change to a mode that is not enabled will not be successful and will result in an interrupt if "invalid mode" interrupts are enabled.

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

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

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

Figure 67. Mode Enable Register (ME_ME)

Table 80. Mode Enable Register (ME_ME) Field Descriptions

Field	Description
RESET_DEST	‘destructive’ RESET mode enable 1 **‘destructive’** RESET mode is enabled **Note:** Destructive resets are where a critical hardware failure, such as an LVI, has occurred. Destructive resets go through the full reset sequence and no context or memory contents can be assumed to be maintained.
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 1 RUN0 mode is enabled

Table 80. Mode Enable Register (ME_ME) Field Descriptions (continued)

Field	Description
DRUN	DRUN mode enable 1 DRUN mode is enabled
SAFE	SAFE mode enable 1 SAFE mode is enabled
TEST	TEST mode enable 0 TEST mode is disabled 1 TEST mode is enabled
RESET_FUNC	'functional' RESET mode enable 1 **'functional'** RESET mode is enabled

8.3.2.4 Interrupt Status Register (ME_IS)

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

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

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

Figure 68. Interrupt Status Register (ME_IS)

This register provides the current interrupt status.

Table 81. Interrupt Status Register (ME_IS) Field Descriptions

Field	Description
I_ICONF_CU	**Invalid mode configuration interrupt (Clock Usage)** — This bit is set during a mode transition if a clock which is required to be on by an enabled peripheral is configured to be turned off. It is cleared by writing a '1' to this bit. 0 No invalid mode configuration (clock usage) interrupt occurred 1 Invalid mode configuration (clock usage) interrupt is pending
I_ICONF	**Invalid mode configuration interrupt** — This bit is set whenever a write operation to ME_<*mode*>_MC registers with invalid mode configuration is attempted. It is cleared by writing a '1' to this bit. 0 No invalid mode configuration interrupt occurred 1 Invalid mode configuration interrupt is pending
I_IMODE	**Invalid mode interrupt** — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a '1' to this bit. Invalid mode interrupts can be caused by one of 5 events: – Mode transition requested while another mode transition is active – The Target mode is not a valid transition from current mode (e.g. DRUN to STOP) – The target mode is not enabled in the ME_ME register – The target mode does not exist – The current mode is SAFE mode If an invalid mode interrupt is flagged, the cause of the interrupt can be determined by looking at the ME_IMTS register (see *Section 8.3.2.6: Invalid Mode Transition Status Register (ME_IMTS)*) 0 No invalid mode interrupt occurred 1 Invalid mode interrupt is pending
I_SAFE	SAFE **mode interrupt** — This bit is set whenever the chip enters SAFE mode on hardware requests generated in the system. It is cleared by writing a '1' to this bit. 0 No SAFE mode interrupt occurred 1 SAFE mode interrupt is pending
I_MTC	**Mode transition complete interrupt** — This bit is set whenever the mode transition process completes (S_MTRANS transits from 1 to 0). It is cleared by writing a '1' to this bit. This mode transition interrupt bit will not be set while entering low-power modes HALT, STOP, or STANDBY. 0 No mode transition complete interrupt occurred 1 Mode transition complete interrupt is pending

8.3.2.5 Interrupt Mask Register (ME_IM)

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

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

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

Figure 69. Interrupt Mask Register (ME_IM)

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

Table 82. Interrupt Mask Register (ME_IM) Field Descriptions

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

8.3.2.6 Invalid Mode Transition Status Register (ME_IMTS)

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

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

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

Figure 70. Invalid Mode Transition Status Register (ME_IMTS)

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

Table 83. 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 chip is in SAFE mode, SAFE event bit is pending and a new mode requested other than RESET/SAFE modes. It is cleared by writing a ‘1’ to this bit. 0 No new mode requested other than RESET/SAFE while SAFE event is pending 1 New mode requested other than RESET/SAFE while SAFE event is pending

8.3.2.7 Debug Mode Transition Status Register (ME_DMTS)

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

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

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

Figure 71. Debug Mode Transition Status Register (ME_DMTS)

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 84. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions

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

Table 84. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
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(1) — This bit is set when any peripheral has been requested to have its clock disabled. It is cleared when all the peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_96_127	Clock Disable Process Pending status for Peripherals 96…127[2] — 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
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64…95[2] — 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[2] — 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[2] — 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

1. Peripheral n corresponds to the ME_PCTLn register. Please refer to *Table 76* for the ME_PCTLn locations actually occupied, which in turn indicates which peripherals are reported in the ME_DMTS register.

8.3.2.8 RESET Mode Configuration Register (ME_RESET_MC)

This register details the mode configuration during reset. See *Table 85* for details.

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

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

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

Figure 72. RESET Mode Configuration Register (ME_RESET_MC)

8.3.2.9 TEST Mode Configuration Register (ME_TEST_MC)

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

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

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

Figure 73. TEST Mode Configuration Register (ME_TEST_MC)

This register configures the system behavior during TEST mode. Please see *Table 85* for details.

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

8.3.2.10 SAFE Mode Configuration Register (ME_SAFE_MC)

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

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

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

Figure 74. SAFE Mode Configuration Register (ME_SAFE_MC)

This register configures the system behavior during SAFE mode. Please see *Table 85* for details.

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

8.3.2.11 DRUN Mode Configuration Register (ME_DRUN_MC)

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

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

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

Figure 75. DRUN Mode Configuration Register (ME_DRUN_MC)

This register configures the system behavior during DRUN mode. Please see *Table 85* for details.

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

Note: The following configuration values are retained through STANDBY mode:

- *CFLAON*
- *DFLAON*
- *FXOSCON*

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

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

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

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

Figure 76. RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC)

This register configures the system behavior during RUN0…3 modes. Please see *Table 85* for details.

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

8.3.2.13 HALT Mode Configuration Register (ME_HALT_MC)

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

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

Figure 77. HALT Mode Configuration Register (ME_HALT_MC)

This register configures the system behavior during HALT mode. Please see *Table 85* for details.

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

8.3.2.14 STOP Mode Configuration Register (ME_STOP_MC)

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

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

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

Figure 78. STOP Mode Configuration Register (ME_STOP_MC)

This register configures system behavior during STOP mode. Please see *Table 85* for details.

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

8.3.2.15 STANDBY Mode Configuration Register (ME_STANDBY_MC)

Address 0xC3FD_C054 Access: User read, Supervisor read/write, Test read/write

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

Figure 79. STANDBY Mode Configuration Register (ME_STANDBY_MC)

This register configures the system behavior during STANDBY mode. Please see *Table 85* for details.

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

Table 85. 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 the pad power sequence driver is disabled, but the state of the output remains functional. 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 **Note:** If the MVRON bit is cleared in STOP or HALT modes, the SRAM is placed into a low power state where it cannot be accessed. The SRAM contents are retained and will be available again after exiting the low power mode.
DFLAON	Data flash power-down control — This bit specifies the operating mode of the data flash after entering this mode. 00 reserved 01 Data flash is in power-down mode 10 reserved 11 Data flash is in normal mode **Note:** If CFLAON ≠ 0b11, then DFLAON cannot be programmed to 0b11 (this effectively becomes reserved). Therefore, the only valid case for DFLAON = 0b11 is when CFLAON is also 0b11. This also affects the order these fields are set and cleared. **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 9.3.1.6: Functional Event Short Sequence Register (RGM_FESS)*).
CFLAON	Code flash power-down control — This bit specifies the operating mode of the code flash after entering this mode. 00 reserved 01 Code flash is in power-down mode 10 Code flash is in low-power mode 11 Code flash is in normal mode

Table 85. Mode Configuration Registers (ME_<*mode*>_MC) field descriptions (continued)

Field	Description
FMPLLON	FMPLL control 0 FMPLL is switched off 1 FMPLL is switched on
FXOSCON	FXOSC (4-40 MHz external oscillator) control 0 FXOSC (4-40 MHz external oscillator) is switched off 1 FXOSC (4-40 MHz external oscillator) is switched on
FIRCON	FIRC control 0 FIRC is switched off 1 FIRC is switched on
SYSCLK	System clock switch control — These bits specify the system clock to be used by the system. 0000 FIRC 0001 FIRC_divided 0010 FXOSC 0011 FXOSC_divided 0100 FMPLL 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled in STOP and TEST modes, reserved in all other modes

8.3.2.16 Peripheral Status Register 0 (ME_PS0)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	S_FlexRay	S_DMA_CH_MUX	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	S_LINFlexD_9	S_LINFlexD_8	S_DSPI7	S_DSPI6	S_DSPI5	S_DSPI4	S_DSPI3	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

Figure 80. Peripheral Status Register 0 (ME_PS0)

This register provides the status of the peripherals. Please see *Table 86* for details.

8.3.2.17 Peripheral Status Register 1 (ME_PS1)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_CANSampler	0	0	S_CTUL	0	S_LINFlexD_7	S_LINFlexD_6	S_LINFlexD_5	S_LINFlexD_4	S_LINFlexD_3	S_LINFlexD_2	S_LINFlexD_1	S_LINFlexD_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	S_I2C	0	0	0	0	0	0	0	0	0	0	S_ADC1	S_ADC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 81. Peripheral Status Register 1 (ME_PS1)

This register provides the status of the peripherals. Please see *Table 86* for details.

8.3.2.18 Peripheral Status Register 2 (ME_PS2)

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

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

Figure 82. Peripheral Status Register 2 (ME_PS2)

This register provides the status of the peripherals. Please refer to *Table 86* for details.

8.3.2.19 Peripheral Status Register 3 (ME_PS3)

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

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

Figure 83. Peripheral Status Register 3 (ME_PS3)

This register provides the status of the peripherals. Please see *Table 86* for details.

Table 86. Peripheral Status Registers (ME_PSn) Field Descriptions

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

8.3.2.20 Run Peripheral Configuration Registers (ME_RUN_PC0…7)

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

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

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

Figure 84. Run Peripheral Configuration Registers (ME_RUN_PC0…7)

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

Table 87. Run Peripheral Configuration Registers (ME_RUN_PC0…7) Field Descriptions

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

8.3.2.21 Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

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

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

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

Figure 85. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

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

Table 88. 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

8.3.2.22 Peripheral Control Registers (ME_PCTL*n*)

These registers select the configurations during run and non-run modes for each peripheral. Please refer to *Table 76* for information on which ME_PCTLn locations are actually occupied. The unoccupied locations contain a read-only byte value of 0x00.

See *Section 8.4.6: Peripheral clock gating*, for details on how to use this register.

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

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

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

Figure 86. Peripheral Control Registers (ME_PCTLn)

Table 89. Peripheral Control Registers (ME_PCTLn) Field Descriptions

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

Peripheral	ME_PCTLn
ADC_0	32
ADC_1	33
CAN sampler	60
CMU	104
CTU	57
DMA_MUX	23
DSPI_0	4
DSPI_1	5
DSPI_2	6
DSPI_3	7
DSPI_4	8
DSPI_5	9
DSPI_6	10

Table 90. Peripheral control registers by peripheral (continued)

Peripheral	ME_PCTLn
DSPI_7	11
eMIOS_0	72
eMIOS_1	73
FlexCAN_0	16
FlexCAN_1	17
FlexCAN_2	18
FlexCAN_3	19
FlexCAN_4	20
FlexCAN_5	21
I2C	44
LINFlexD_0	48
LINFlexD_1	49
LINFlexD_2	50
LINFlexD_3	51
LINFlexD_4	52
LINFlexD_5	53
LINFlexD_6	54
LINFlexD_7	55
LINFlexD_8	12
LINFlexD_9	13
PIT_RTI	92
RTC/API	91
SIUL	68
WKPU	69

8.4 Functional description

8.4.1 Mode transition request

Automatic mode transitions are handled by hardware in the case of special events such as low power mode exit as well as handling fault conditions. Software initiated mode transitions are handled by writing to the mode control (ME_MCTL) register which must be written twice:

1. Write the requested mode to the TARGET_MODE field and the key (0x5AF0) to the KEY field.
2. On the second write the TARGET_MODE value is the same but the KEY is inverted (0xA50F).

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. It is valid to enter and re-enter the same mode (for example, DRUN → DRUN) to latch any changes in configuration. As soon as the mode request is accepted as valid, the S_MTRANS bit is set and held until 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.

Note: A mode re-entry is required to latch changes to any of the Mode Control, Mode Enable, Mode Configuration, ME_RUN_PCn, ME_LP_PCn and ME_PCTLn registers.

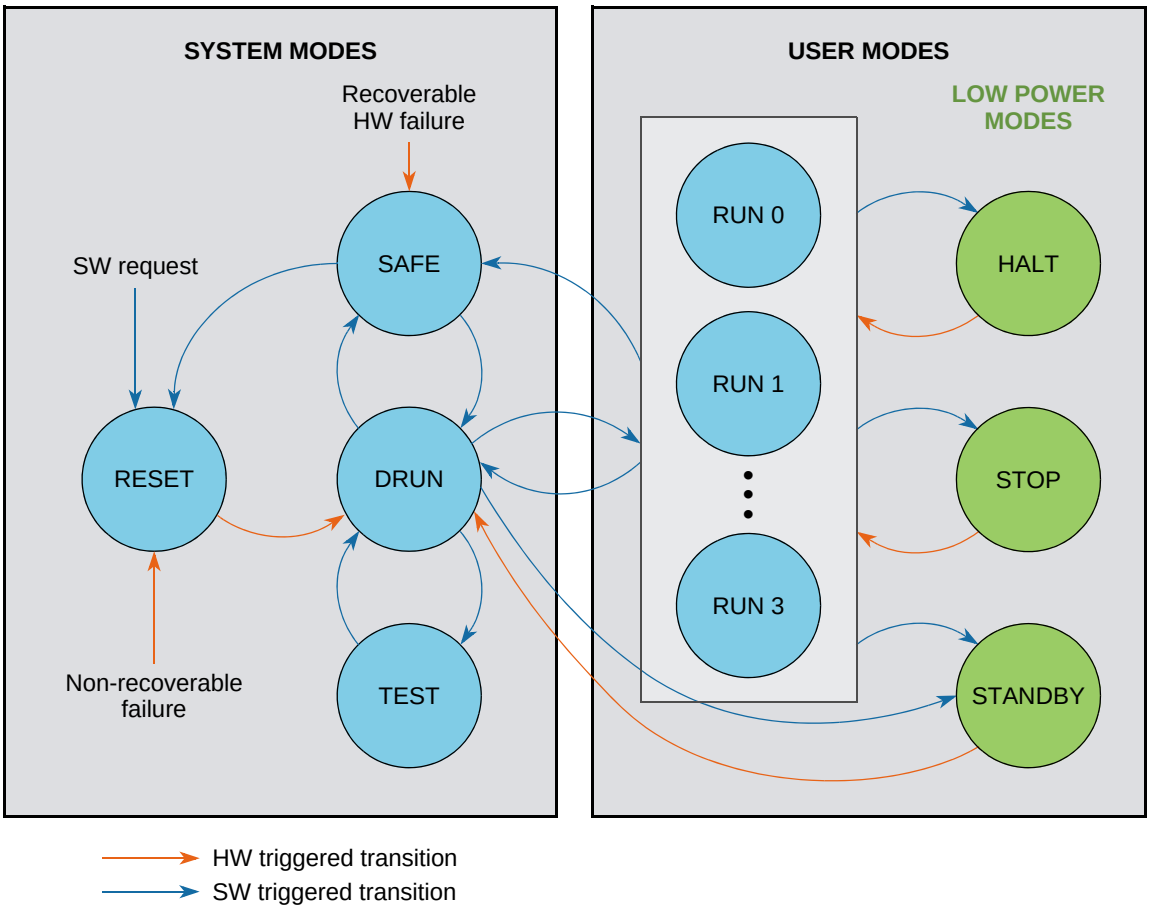


Figure 87. MC_ME mode diagram

8.4.2 Mode details

8.4.2.1 RESET mode

The microcontroller 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 either “0000” for a ‘functional’ reset or “1111” for a ‘destructive’ reset
- from any mode due to a system reset by the MC_RGM because of some non-recoverable hardware failure in the system (see the MC_RGM chapter for details)

Transition to this mode is instantaneous, and the system remains in this mode until the reset sequence is finished. The mode configuration information for this mode is provided by the ME_RESET_MC register. This mode has a pre-defined configuration, and the FIRC is selected as the system clock. All power domains are made active in this mode.

8.4.2.2 DRUN mode

The microcontroller 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 (for example, 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 CFlash and DFlash, 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 FIRC as the system clock.

DRUN mode:

- Is the initial mode that user code is executed in
- Provides the gateway to RUN[0..3] modes
- Can be used to jump into and recover from STANDBY mode

as the initial mode that user code is executed in.

When DRUN 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.

In DRUN mode, all power domains are active.

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

8.4.2.3 SAFE mode

The microcontroller enters this mode on the following events:

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

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

As soon as any of the above events has occurred, a SAFE mode transition request is generated. The mode configuration information for this mode is provided by the ME_SAFE_MC register. This mode has a pre-defined configuration, and the FIRC 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 or new requests during the mode transition. No new mode request made during a transition to the SAFE mode will cause an invalid mode interrupt.

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

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

This mode is intended to be used by software

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

If the outputs of the system I/Os need to be forced to a high impedance state upon entering this mode, the PDO bit of the ME_SAFE_MC register should be set. In this case, the pads' power sequence driver cell is also disabled. The input levels remain unchanged.

8.4.2.4 TEST mode

The microcontroller enters this mode on the following events:

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

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

This mode is intended to be used by software

- to execute software test routines

In TEST mode, all power domains are active.

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

8.4.2.5 RUN0…3 modes

The microcontroller enters one of RUN0...3 modes on the following events:

- from the DRUN, SAFE, or another RUN0…3 mode when the TARGET_MODE bit field of the ME_MCTL register is written with "0100…0111"
- from the HALT mode due to an interrupt event
- from the STOP mode due to an interrupt or wakeup event

As soon as any of the above events has occurred, a RUN0…3 mode transition request is generated. The mode configuration information for these modes is provided by the ME_RUN0…3_MC registers. In these modes, the CFlash and DFlash, 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
- To allow entry into STOP and HALT modes

In RUN0..3 modes, all power domains are active.

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

8.4.2.6 HALT mode

The chip enters this mode on the following events:

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

As soon as any of the above events has occurred, a HALT mode transition request is generated. The mode configuration information for this mode is provided by ME_HALT_MC register. The main voltage regulator and the CFlash and DFlash can be put in low-power or power-down mode as needed. If there is a HALT mode request while an interrupt request is active, the transition to HALT is aborted with the resultant mode being the current mode, SAFE (on SAFE mode request), or DRUN (on reset), and an invalid mode interrupt is not generated.

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

- the core clock frozen
- only a few peripherals running

and to be used by software

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

In HALT mode, all power domains are active. However, if the ME_HALT_MC[MVRON] bit is cleared, the SRAM is placed into a low power state where it cannot be accessed. The SRAM contents are retained and will be available again after exiting HALT mode.

Note: It is good practice for software to ensure that the ME_GS[S_MTRANS] bit has been cleared on HALT mode exit to ensure that the previous RUN0…3 mode configuration has been fully restored before executing critical code.

8.4.2.7 STOP mode

The microcontroller enters this mode on the following events:

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

As soon as any of the above events has occurred, 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 main voltage regulator and the CFlash and DFlash 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 transition to STOP mode is aborted with the resultant mode being one of the following:
  - The current mode
  - SAFE (on SAFE mode request)
  - DRUN (on reset)
- An invalid mode interrupt is not generated.

If any interrupt is pending, the mode is not entered until all interrupt flags are cleared if the system clock is still available.

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 system 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 (for example, to 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 and thus preserve the chip status. When exiting the STOP mode, the FIRC clock is selected as the system clock until the target clock is available.

In STOP mode, all power domains are active. However, if the ME_STOP_MC[MVRON] bit is cleared, the SRAM is placed into a low power state where it cannot be accessed. The SRAM contents are retained and will be available again after exiting STOP mode.

Note: It is good practice for software to ensure that the ME_GS[S_MTRANS] bit has been cleared on STOP mode exit to ensure that the previous RUN0…3 mode configuration has been fully restored before executing critical code.

8.4.2.8 STANDBY mode

The microcontroller enters STANDBY 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 microcontroller. The only parts of the microcontroller that are still powered during STANDBY 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, SSCM, and VREG. The FIRC can be optionally switched off. This is the lowest power consumption mode possible.

This mode is intended:

- As an extreme low-power mode with everything powered down apart from the necessary circuitry to allow device wakeup
- To be used by software to remain in the lowest power consumption state with no requirement to wake up quickly

The exit sequence of this mode is similar to the reset sequence, and the resets to all but power domain #0 are asserted with timing that is the same as reset PHASE1 through PHASE3. However, in addition to booting from the default location, the microcontroller can also be configured to boot from the backup RAM (see the RGM_STDBY register description in the MC_RGM chapter for details). In the case of booting from backup RAM, it is also

possible to keep the CFlash and DFlash disabled by writing “01” to the CFLAON and DFLAON fields in the ME_DRUN_MC register prior to STANDBY entry.

If there is a STANDBY mode request while any wakeup event is active, the microcontroller will not enter STANDBY mode.

In STANDBY mode power domain 1 is disabled. Power domains 2 and 3 can be selectively configured to enable additional SRAM as defined in *Table 108: RAM configurations in modes*.

Note: Communications protocols that do not support hardware handshaking (for example, FlexRay) need to be manually disabled before low power mode entry.

Note: Initialization code in RAM should not enable WKPU clock through configuring PCTL69 until the NMI exception handler is installed. If WKPU is enabled before installing the handler, the software might jump to an illegal code.

8.4.3 Mode transition process

The process of mode transition follows the following steps in a pre-defined manner depending on the current mode and the requested target mode. In many cases of mode transition, not all steps need to be executed based on the mode control information, and some steps may not be applicable according to the mode definition itself.

8.4.3.1 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. See *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. It can be generated either by software via the ME_MCTL register from all software running modes including DRUN, RUN0…3, and TEST or by the MC_RGM after the detection of system hardware failures, which may occur in any mode.

8.4.3.2 Target mode configuration loading

On completion of the *Section 8.4.3.1: Target mode request* step, the resources for that mode (Pad, VREG, CFlash, DFlash, and clock configuration) are read from the ME_<target mode>_MC register and configured accordingly.

Table 91 shows the resources and configuration that are available for each operating mode. Shaded cells indicate that the configuration is hard wired and cannot be changed.

Table 91. MC_ME resource control overview(1)

Mode	PDO	Main VREG	Data flash memory	Code flash memory	FMPLL	FXOSC	FIRC	SYSCLK
RESET	Off	On	On	On	Off	Off	On	FIRC
TEST	Off	On	On	On	Off	Off	On	Disabled
SAFE	On	On	On	On	Off	Off	On	FIRC
DRUN	Off	On	On	On	Off	Off	On	FIRC
RUN0..3	Off	On	On	On	Off	Off	On	FIRC
HALT	Off	On	On	Low power	Off	Off	On	FIRC
STOP	Off	On	Off	Off	Off	Off	On	FIRC
STANDBY	On	Off	Off	Off	Off	Off	On	Disabled

1. Shaded cells represent configurations that cannot be changed in that mode. Unshaded cells show the default value that can be modified as required.

8.4.3.3 Peripheral clocks disable

On completion of the *Section 8.4.3.1: Target mode request* step, the MC_ME requests each peripheral to enter its stop mode if the peripheral is configured to be disabled in the target mode based on the configuration of the peripheral PCTL register and the referenced ME_RUN_PC0..7 or ME_LP_PC0..7 registers.

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

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

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

Please refer to *Section 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 microcontroller enters the SAFE mode.

8.4.3.4 Processor low-power mode entry

If, on completion of the *Section 8.4.3.3: Peripheral clocks disable* step, 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 *Section 8.4.3.3: Peripheral clocks disable* step, 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.

8.4.3.5 Processor and System Memory Clock Disable

If, on completion of the *Section 8.4.3.4: Processor low-power mode entry* step, 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 memory are unaffected while transitioning between software running modes such as DRUN, RUN0…3, and SAFE.

Caution: Clocks to the whole microcontroller including the processor and system memory can be disabled in TEST mode.

8.4.3.6 Clock Sources (Main Voltage Regulator Independent) Switch-On

On completion of the *Section 8.4.3.4: Processor low-power mode entry* step, the MC_ME switches on all clock sources, which do not need the main voltage regulator to be on, based on the <clock source>ON bits of the ME_<current mode>_MC and ME_<target mode>_MC registers. The following clock sources are switched on at this step:

- FXOSC
- FIRC

Note: Clock sources which need the main voltage regulator to be stable are 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 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.

8.4.3.7 Main Voltage Regulator Switch-On

On completion of the *Section 8.4.3.1: Target mode request* step, 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 FIRC is switched on regardless of the target mode configuration, as the main voltage regulator requires the FIRC 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).

8.4.3.8 Flash memory modules switch-on

On completion of the *Section 8.4.3.7: Main Voltage Regulator Switch-On* step, if one or more of the flashes needs to be switched to normal mode from its low-power or power-down mode based on the CFLAON and DFLAON bit fields of the ME_<current mode>_MC and ME_<target mode>_MC registers, the MC_ME requests the flash to exit from its low-

power/power-down mode. When the flashes are available for access, the S_CFLA and S_DFLA bit fields of the ME_GS register are updated to “11” by hardware.

If the main regulator is also off in chip low-power modes, then during the exit sequence, the flash is kept in its low-power state and is switched on only when the *Section 8.4.3.7: Main Voltage Regulator Switch-On* process has completed.

Caution: It is illegal to switch the CFlash directly from low-power mode to power-down mode or from power-down mode to low-power mode. The MC_ME, however, does not prevent this nor does it flag it.

8.4.3.9 Clock Sources (Main Voltage Regulator Dependent) Switch-On

On completion of the *Section 8.4.3.6: Clock Sources (Main Voltage Regulator Independent) Switch-On* and *Section 8.4.3.7: Main Voltage Regulator Switch-On*, the MC_ME controls all clock sources, which need the main voltage regulator to be on, based on the <clock source>ON bits of the ME_<current mode>_MC and ME_<target mode>_MC registers. The following clock sources are switched on at this step:

- FMPLL

8.4.3.10 Pad Outputs-On

On completion of the *Section 8.4.3.7: Main Voltage Regulator Switch-On* step, if the PDO bit of the ME_<target mode>_MC register is cleared, then

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

8.4.3.11 Peripheral clocks enable

Based on the current and target microcontroller modes, the peripheral configuration registers ME_RUN_PC0…7, ME_LP_PC0…7, and the peripheral control registers ME_PCTLn, the MC_ME enables the clocks for selected modules as required. This step is executed only after the *Section 8.4.3.7: 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. At that time, the MC_ME may assert the peripheral clock enables of the peripherals residing in those power domains.

8.4.3.12 Processor and Memory Clock Enable

If the mode transition is from either HALT or STOP to RUN0…3, the clocks to the processor and system memory are enabled. The process of enabling these clocks is executed only after the *Section 8.4.3.8: Flash memory modules switch-on* process is completed.

8.4.3.13 Processor low-power mode exit

If the mode transition is from 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 *Section 8.4.3.12: Processor and Memory Clock Enable* process is completed.

8.4.3.14 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 FIRC takes effect only after the S_FIRC bit of the ME_GS register is set by hardware (that is, the FIRC has stabilized).
- The target clock configuration for the FIRC_divided takes effect only after the S_FIRC bit of the ME_GS register is set by hardware (that is, the FIRC has stabilized).
- The target clock configuration for the FXOSC takes effect only after the S_FXOSC bit of the ME_GS register is set by hardware (that is, the FXOSC (4-40 MHz external oscillator) has stabilized).
- The target clock configuration for the FXOSC_divided takes effect only after the S_FXOSC bit of the ME_GS register is set by hardware (that is, the FXOSC (4-40 MHz external oscillator) has stabilized).
- The target clock configuration for the FMPLL takes effect only after the S_FMPLL bit of the ME_GS register is set by hardware (that is, the FMPLL 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 *Section 8.4.3.3: Peripheral clocks disable* process has completed in order not to change the system clock frequency before peripherals close their internal activities

Table 92 shows the system clock selection options available in each operating mode. The default configuration is shown. Shading indicates that the SYSCLK cannot be changed from or to this option. A checkmark represents valid SYSCLK settings.

Table 92. MC_ME system clock selection overview

Mode	FIRC	FIRC divided	FXOSC	FXOSC divided	FMPLL	Disabled
RESET	Default	x	x	x	x	x
TEST	Default	√	√	√	√	√
SAFE	Default	x	x	x	x	x
DRUN	Default	√	√	√	√	x
RUN0..3	Default	√	√	√	√	x
HALT	Default	√	√	√	√	x
STOP	Default	√	√	√	√	√
STANDBY	x	x	x	x	x	Default

8.4.3.15 Power Domain #2…3 Switch-Off

Based on the chip mode and the MC_PCU's power configuration registers PCU_PCONF2…3, the power domains #2 to 3 are controlled by the MC_PCU.

If a mode change translates to a power-down of a power domain, then the MC_PCU starts the power-down sequence. The MC_PCU acknowledges the completion of the power-down sequences 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 *Section 8.4.3.3: Peripheral clocks disable* process has completed.

8.4.3.16 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 *Section 8.4.3.3: Peripheral clocks disable* process has completed.

8.4.3.17 Clock Sources Switch-Off

Based on the chip mode and the <clock source>ON bits of the ME_<mode>_MC registers, if a given clock source is to be switched off, the MC_ME:

- Requests the clock source to power down
- Updates its availability status bit S_<clock source> of the ME_GS register to '0'

This step is executed only after the *Section 8.4.3.14: System clock switching* process has completed.

8.4.3.18 Flash Switch-Off

Based on the CFLAON and DFLAON bit fields of the ME_<current mode>_MC and ME_<target mode>_MC registers, if any of the flashes is to be put in its low-power or power-down mode, the MC_ME:

- Requests the flash memory to enter the corresponding power mode
- Waits for the flash memory to acknowledge

The exact power mode status of the flashes is updated in the S_CFLA and S_DFLA bit fields of the ME_GS register. This step is executed only when the *Section 8.4.3.5: Processor and System Memory Clock Disable* process has completed.

8.4.3.19 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:

- *Section 8.4.3.17: Clock Sources Switch-Off*
- *Section 8.4.3.18: Flash Switch-Off*

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 FIRC, the FIRC 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.

8.4.3.20 Current mode update

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

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

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

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

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

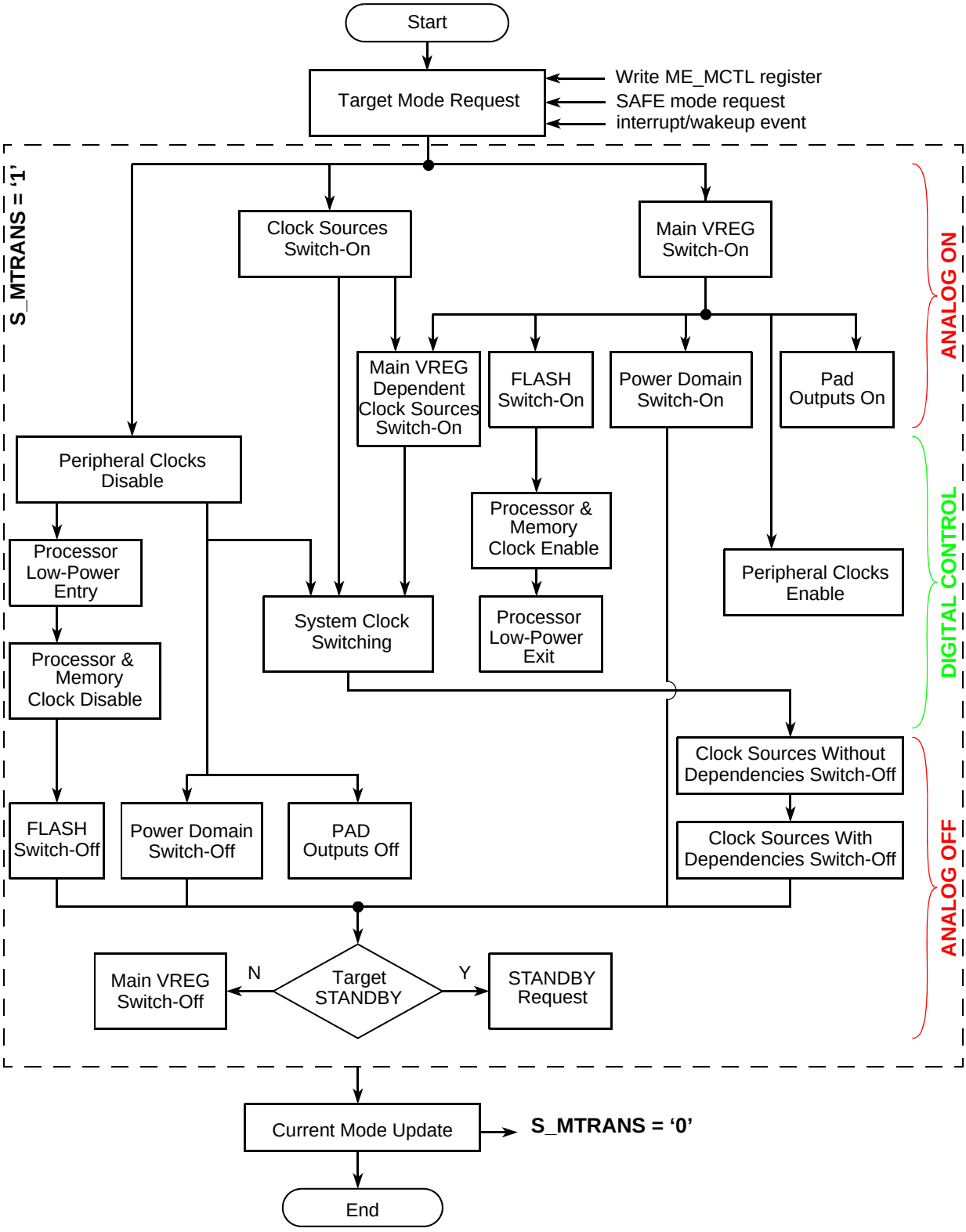


Figure 88. MC_ME transition diagram

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.

- If the FIRC is selected as the system clock, FIRC must be on.
- If the FIRC_divided clock is selected as the system clock, FIRC must be on.
- If the FXOSC clock is selected as the system clock, FXOSC must be on.
- If the FXOSC_divided clock is selected as the system clock, FXOSC must be on.
- If the FMPLL clock is selected as the system clock, PLL must be on.

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

- Configuration “00” for the CFLAON bit field is reserved.
- Configuration “00” for the DFLAON bit field is reserved.
- Configuration “10” for the DFLAON bit field is reserved.
- Configuration “11” for the DFLAON bit field is reserved if the CFLAON bit field is not “11”.
- MVREG must be on if any of the following is active:
  - 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 chip can exit only via a system reset.

8.4.5 Mode transition interrupts

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

8.4.5.1 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 the ME_IM register is ‘1’.

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

8.4.5.2 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 (that is, 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 modes are disabled in the ME_ME register, their respective configurations are considered reserved, and any attempt to change to a disabled mode by writing to the ME_MCTL will result in an invalid mode transition request. When such a disabled mode is requested, the S_DMA bit of the ME_IMTS register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME_MCTL register.
- If the target mode is not a valid mode with respect to the current mode, the mode request illegal status bit S_MRI of the ME_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME_MCTL register. Otherwise, the write operation is ignored.
- If further new mode requests occur while a mode transition is in progress (the S_MTRANS bit of the ME_GS register is ‘1’), the mode transition illegal status bit S_MTI of the ME_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME_MCTL register. Otherwise, the write operation is ignored.

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

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

- A new request is allowed to enter the RESET or SAFE mode irrespective of the mode transition status.
- As the exit of 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 chip modes, software can abort a mode transition by requesting the parent mode if, for example, the mode transition has not completed after a software determined ‘reasonable’ amount of time for whatever reason. The parent mode is the chip mode before a valid mode request was made.
- Self-transition requests (for example, RUN0 → RUN0) are not considered as invalid even when the mode transition process is active (that is, 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 (that is, all status bits of the ME_GS register match with configuration bits in the ME_<mode>_MC register), then software can only request the SAFE or RESET mode. It is not possible to request any other mode or to go back to the low-power mode again.

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

8.4.5.3 SAFE mode transition interrupt

Whenever the system enters 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 (that is, programming of ME_MCTL register).

8.4.5.4 Mode transition complete interrupt

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

8.4.6 Peripheral clock gating

Each peripheral has an associated Peripheral Control Register (ME_PCTLn), as described in *Section 8.3.2.22: Peripheral Control Registers (ME_PCTLn)*. The RUN_CFG and LP_CFG fields within the ME_PCTL registers are used as determine whether the peripheral is enabled or disabled (clock gated) in run modes and low power modes as described in this section.

The ME_PCTLn[RUN_CFG] field references one of 8 Run Peripheral Configuration (ME_RUN_PC[0..7]) registers. Each of these identical registers has a bit for each run mode (RUN0..3, DRUN, SAFE and TEST) that can be set to indicate that the associated group of peripherals (referenced by ME_PCTLn[RUN_CFG]) is either enabled or disabled in this mode.

Similarly, the ME_PCTLn[LP_CFG] field references one of the 8 Low Power Peripheral Configuration Registers (ME_LP_PC[0..7]). As with the run mode registers, there is a bit for each low power mode (STANDBY, STOP and HALT) that determines whether the referenced peripheral group is active in each mode.

This highly flexible configuration allows 8 groups of peripherals (one per PC[0..7] register) to be established that are available in different modes. For example, one group of peripherals could be available in RUN1 mode whereas another group was active in RUN2 mode. By simply switching modes, the new set of peripherals is available.

Figure 89 shows this scheme in more detail.

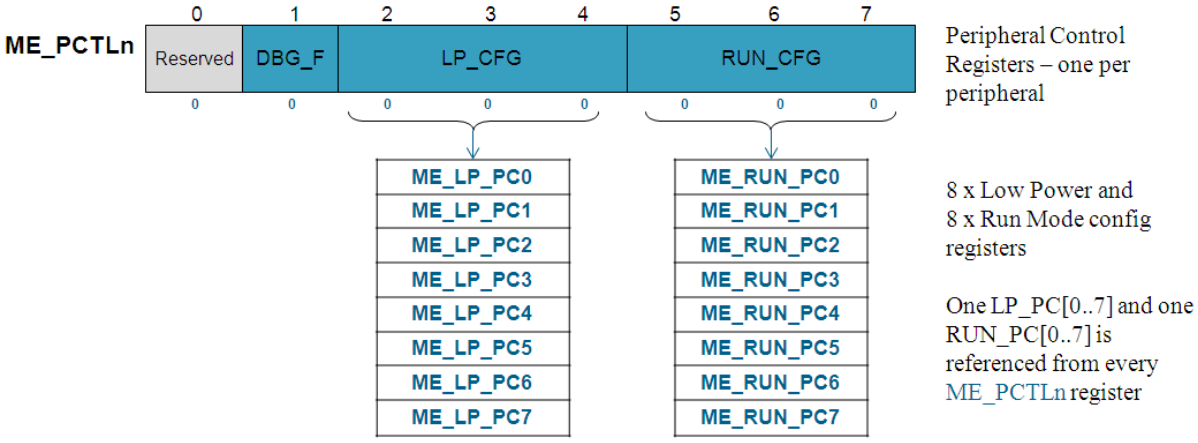


Figure 89. Interaction between ME_PCTLn, ME_LPn, and ME_RUN_PCn

Any modifications to the ME_RUN_PC[0..7], ME_LP_PC[0..7] or PCTLn registers will not take effect until a mode transition request has been registered.

By default, each ME_PCTLn register has a reset value of 0x00. This means that each peripheral is associated with ME_RUN_PC[0] and ME_LP_PC[0] by default. Therefore, a fast way to initialize all of the peripherals in the desired modes is simply to set the desired mode bits within ME_RUN_PC[0] and ME_LP_PC[0].

The ME_PCTLn registers also have a DBG_F bit. This bit determines the peripheral behaviour when the microcontroller enters debug mode. If the DBG_F bit is set, the peripheral is clock gated in debug mode. If the bit is cleared, then the availability of the peripheral in debug mode depends on the current mode and the settings of the ME_RUN_PC[0..7] and ME_LP_PC[0..7] fields.

8.4.7 Application example

Figure 90 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

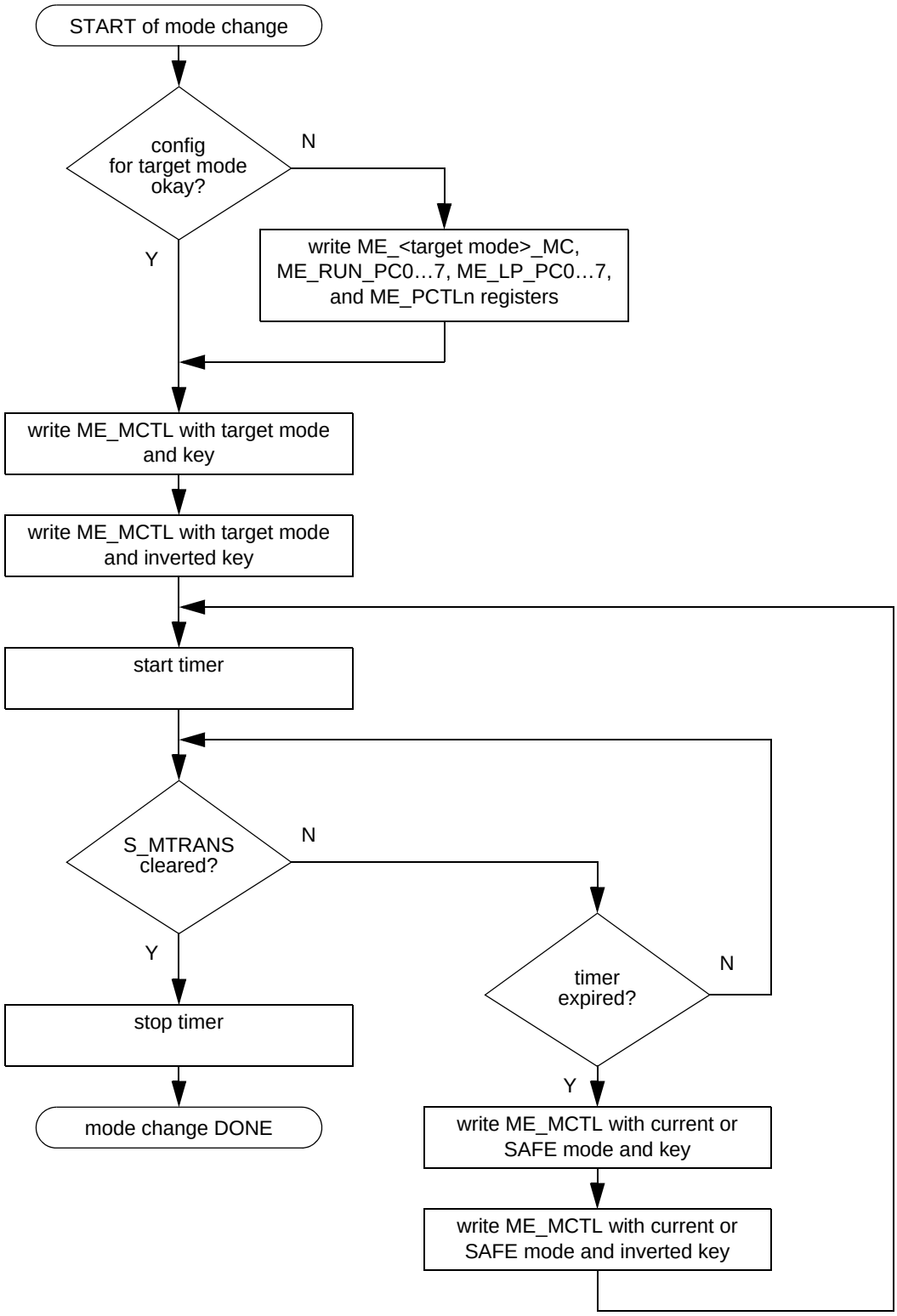


Figure 90. MC_ME Application example flow diagram

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

Figure 91 depicts the MC_RGM block diagram.

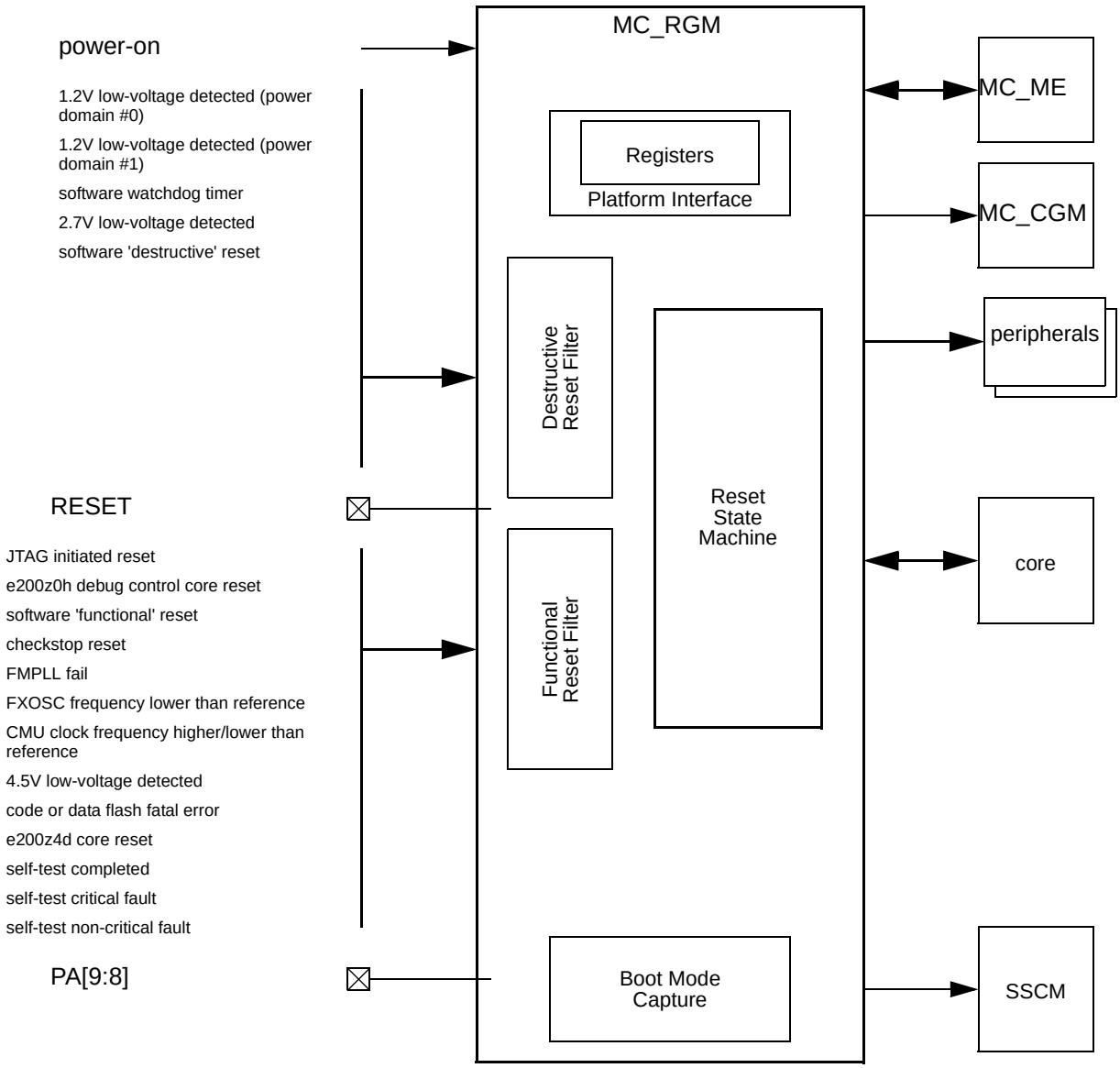


Figure 91. MC_RGM block diagram

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
- short reset sequence configuration
- bidirectional reset behavior configuration
- selection of alternate boot via the backup RAM on STANDBY mode exit
- boot mode capture on RESET deassertion

9.1.3 Reset sources

The different reset sources are organized into two families: ‘destructive’ and ‘functional’.

- A ‘destructive’ reset source is associated with an event related to a critical - usually hardware - error or dysfunction. When a ‘destructive’ reset event occurs, the full reset sequence is applied to the chip starting from PHASE0. This resets the full chip ensuring a safe start-up state for both digital and analog modules, and the memory content must be considered to be unknown. ‘Destructive’ resets are
  - power-on reset
  - 1.2V low-voltage detected (power domain #0)
  - 1.2V low-voltage detected (power domain #1)
  - software watchdog timer
  - 2.7V low-voltage detected
  - software 'destructive' reset
- A ‘functional’ reset source is associated with an event related to a less-critical - usually non-hardware - error or dysfunction. When a ‘functional’ reset event occurs, a partial reset sequence is applied to the chip starting from PHASE1. In this case, most digital modules are reset normally, and the memory content must be considered to be unknown, while the state of analog modules or specific digital modules (e.g., debug modules, flash modules) is preserved. ‘Functional’ resets are
  - external reset
  - JTAG initiated reset
  - e200z0h debug control core reset
  - software 'functional' reset
  - checkstop reset
  - FMPLL fail
  - FXOSC frequency lower than reference
  - CMU clock frequency higher/lower than reference
  - 4.5V low-voltage detected
  - code or data flash fatal error
  - e200z4d core reset
  - self-test completed
  - self-test critical fault
  - self-test non-critical fault

When a reset is triggered, the MC_RGM state machine is activated and proceeds through the different phases (that is, PHASEn states). Each phase is associated with a particular chip reset being provided to the system. A phase is completed when all corresponding phase completion gates from either the system or internal to the MC_RGM are acknowledged. The chip reset associated with the phase is then released, and the state machine proceeds to the next phase up to entering the IDLE phase. During this entire process, the MC_ME state machine is held in RESET mode. Only at the end of the reset sequence, when the IDLE phase is reached, does the MC_ME enter the DRUN mode.

Alternatively, it is possible for software to configure some reset source events to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt issued to the core (see *Section 9.3.1.3: Functional Event Reset Disable Register (RGM_FERD)* and *Section 9.3.1.5: 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[9:8].

9.3 Memory map and register definition

Table 93. MC_RGM register description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_4000	RGM_FES	Functional Event Status	half-word	read	read/write(1)	read/write(1)	*on page 284*
0xC3FE_4002	RGM_DES	Destructive Event Status	half-word	read	read/write(1)	read/write(1)	*on page 286*
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	half-word	read	read/write(2)	read/write(2)	*on page 287*
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	half-word	read	read	read	*on page 288*
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	half-word	read	read/write	read/write	*on page 289*
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	half-word	read	read/write	read/write	*on page 290*
0xC3FE_401A	RGM_STDBY	STANDBY Reset Sequence	half-word	read	read/write	read/write	*on page 292*
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	half-word	read	read/write	read/write	*on page 293*

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

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

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

Table 94. MC_RGM memory map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4000	RGM_FES / RGM_DES	R	F_EXR	F_ST_NCF	F_ST_CF	F_ST_DONE		F_Z4CORE		F_FLASH	F_LVD45	F_CMU_FHL	F_CMU_OLR	F_FMPLL	F_CHKSTOP	F_SOFT_FUNC	F_Z0CORE	F_JTAG
		W	w1c	w1c	w1c	w1c		w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	F_SOFT_DEST											F_LVD27	F_SWT	F_LVD12_PD1	F_LVD12_PD0
		W	w1c	w1c											w1c	w1c	w1c	w1c
0xC3FE_4004	RGM_FERD / RGM_DERD	R	D_EXR	D_ST_NCF	D_ST_CF	D_ST_DONE		D_Z4CORE		D_FLASH	D_LVD45	D_CMU_FHL	D_CMU_OLR	D_FMPLL	D_CHKSTOP	D_SOFT_FUNC	D_Z0CORE	D_JTAG
		W																
		R	0	D_SOFT_DEST											D_LVD27	D_SWT	D_LVD12_PD1	D_LVD12_PD0
		W																
0xC3FE_4008 … 0xC3FE_400C	reserved																	
0xC3FE_4010	RGM_FEAR	R		AR_ST_NCF	AR_ST_CF			AR_Z4CORE			AR_LVD45	AR_CMU_FHL	AR_CMU_OLR	AR_FMPLL			AR_Z0CORE	AR_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 94. MC_RGM memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
		W																
0xC3FE_4014	reserved																	
0xC3FE_4018	RGM_FESS / RGM_STDBY	R	SS_EXR			SS_ST_DONE		SS_Z4CORE		SS_FLASH	SS_LVD45	SS_CMU_FHL	SS_CMU_OLR	SS_FMPLL	SS_CHKSTOP	SS_SOFT_FUNC	SS_Z0CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	SB_CPU	BOOT_FROM_BKP_RAM	0	0	0	0	0	0	0
		W																
0xC3FE_401C	RGM_FBRE	R	BE_EXR			BE_ST_DONE		BE_Z4CORE		BE_FLASH	BE_LVD45	BE_CMU_FHL	BE_CMU_OLR	BE_FMPLL	BE_CHKSTOP	BE_SOFT_FUNC	BE_Z0CORE	BE_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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_DES[8:15] register bits may be accessed as a word at address 0xC3FE_4000, as a half-word at address 0xC3FE_4002, or as a byte at address 0xC3FE_4003.

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

9.3.1.1 Functional Event Status Register (RGM_FES)

Address 0xC3FE_4000 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	F_ST_NCF	F_ST_CF	F_ST_DONE		F_Z4CORE		F_FLASH	F_LVD45	F_CMU_FHL	F_CMU_OLR	F_FMPLL	F_CHKSTOP	F_SOFT_FUNC	F_Z0CORE	F_JTAG
W	w1c	w1c	w1c	w1c		w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 92. Functional Event Status Register (RGM_FES)

This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. It can be accessed in read only in user mode. Register bits are cleared on write '1' if the triggering event has already been cleared at the source.

Note: If a 'functional' reset source is configured to generate a SAFE mode request or an interrupt request, software needs to clear the event in the source module at least three system clock cycles before it clears the associated RGM_FES status bit in order to avoid multiple SAFE mode requests or interrupts for the same event. In order to avoid having to count cycles, it is good practice for software to check whether the RGM_FES has been properly cleared, and if not, clear it again.

Table 95. 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_ST_NCF	Flag for self-test non-critical fault 0 No self-test non-critical fault event has occurred since either the last clear or the last destructive reset assertion 1 A self-test non-critical fault event has occurred
F_ST_CF	Flag for self-test critical fault 0 No self-test critical fault event has occurred since either the last clear or the last destructive reset assertion 1 A self-test critical fault event has occurred
F_ST_DONE	Flag for self-test completed 0 No self-test completed event has occurred since either the last clear or the last destructive reset assertion 1 A self-test completed event has occurred
F_Z4CORE	Flag for e200z4d core reset 0 No e200z4d core reset event has occurred since either the last clear or the last destructive reset assertion 1 A e200z4d core reset event has occurred

Table 95. Functional Event Status Register (RGM_FES) Field Descriptions (continued)

Field	Description
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 4.5V low-voltage detected 0 No 4.5V low-voltage detected event has occurred since either the last clear or the last destructive reset assertion 1 A 4.5V low-voltage detected event has occurred
F_CMU_FHL	Flag for CMU clock frequency higher/lower than reference 0 CMU indicated that FMPLL frequency in range if PLL monitoring is enabled in CMU 1 CMU indicated that FMPLL frequency out of range
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
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_FUNC	Flag for software 'functional' reset 0 No software 'functional' reset event has occurred since either the last clear or the last destructive reset assertion 1 A software 'functional' reset event has occurred
F_Z0CORE	Flag for e200z0h debug control core reset 0 No e200z0h debug control core reset event has occurred since either the last clear or the last destructive reset assertion 1 A e200z0h 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

9.3.1.2 Destructive Event Status Register (RGM_DES)

Address 0xC3FE_4002 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_POR	F_SOFT_DEST											F_LVD27	F_SWT	F_LVD12_PD1	F_LVD12_PD0
W	w1c	w1c											w1c	w1c	w1c	w1c
POR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 93. Destructive Event Status Register (RGM_DES)

This register contains the status of the last asserted destructive reset sources. It can be accessed in read/write on either supervisor mode or test mode. It can be accessed in read only in user mode. Register bits are cleared on write ‘1’.

Table 96. Destructive Event Status Register (RGM_DES) field descriptions

Field	Description
F_POR	Flag for Power-On reset 0 No power-on event has occurred since the last clear 1 A power-on event has occurred
F_SOFT_DEST	Flag for software 'destructive' reset 0 No software 'destructive' reset event has occurred since either the last clear or the last power-on reset assertion 1 A software 'destructive' reset event has occurred
F_LVD27	Flag for 2.7V low-voltage detected 0 No 2.7V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V 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 1.2V low-voltage detected (power domain #1) 0 No 1.2V low-voltage detected (power domain #1) event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2V low-voltage detected (power domain #1) event has occurred
F_LVD12_PD0	Flag for 1.2V low-voltage detected (power domain #0) 0 No 1.2V low-voltage detected (power domain #0) event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2V low-voltage detected (power domain #0) event has occurred

Note: The F_POR flag is also set when a low-voltage is detected on the 1.2 V supply, even if the low voltage is detected after power-on has completed.

Note: The F_LVD27 flag may still have the value '0' after a dip has occurred on the 2.7 V supply during a non-monotonic power-on sequence. The F_POR flag will, however, still be set in this case as expected after each power-on sequence.

9.3.1.3 Functional Event Reset Disable Register (RGM_FERD)

Address 0xC3FE_4004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D_EXR	D_ST_NCF	D_ST_CF	D_ST_DONE		D_Z4CORE		D_FLASH	D_LVD45	D_CMU_FHL	D_CMU_OLR	D_FMPLL	D_CHKSTOP	D_SOFT_FUNC	D_Z0CORE	D_JTAG
W																
POR	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 94. Functional Event Reset Disable Register (RGM_FERD)

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 9.3.1.5: Functional Event Alternate Request Register (RGM_FEAR)*). Some fields are read-only, and their POR value and corresponding behavior cannot be changed. 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.

Caution: It is important to clear the RGM_FES register before setting any of the bits in the RGM_FERD register to ‘1’. Otherwise a redundant SAFE mode request or interrupt request may occur.

Table 97. Functional Event Reset Disable Register (RGM_FERD) field descriptions

Field	Description
D_EXR	**Disable External Reset** 0 An external reset event triggers a reset sequence
D_ST_NCF	**Disable self-test non-critical fault** 1 A self-test non-critical fault event generates an interrupt request
D_ST_CF	**Disable self-test critical fault** 1 A self-test critical fault event generates a SAFE mode request
D_ST_DONE	**Disable self-test completed** 0 A self-test completed event triggers a reset sequence
D_Z4CORE	**Disable e200z4d core reset** 0 A e200z4d core reset event triggers a reset sequence 1 A e200z4d core reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_Z4CORE
D_FLASH	**Disable code or data flash fatal error** 0 A code or data flash fatal error event triggers a reset sequence

Table 97. Functional Event Reset Disable Register (RGM_FERD) field descriptions (continued)

Field	Description
D_LVD45	**Disable 4.5V low-voltage detected** 0 A 4.5V low-voltage detected event triggers a reset sequence 1 A 4.5V low-voltage detected event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_LVD45
D_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
D_CHKSTOP	**Disable checkstop reset** 0 A checkstop reset event triggers a reset sequence
D_SOFT_FUNC	**Disable software 'functional' reset** 0 A software 'functional' reset event triggers a reset sequence
D_Z0CORE	**Disable e200z0h debug control core reset** 0 A e200z0h debug control core reset event triggers a reset sequence 1 A e200z0h debug control core reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_Z0CORE
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

9.3.1.4 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	D_SOFT_DEST											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

Figure 95. Destructive Event Reset Disable Register (RGM_DERD)

This register provides dedicated bits to disable particular destructive reset sources.

Table 98. Destructive Event Reset Disable Register (RGM_DERD) field descriptions

Field	Description
D_SOFT_DEST	**Disable software 'destructive' reset** 0 A software 'destructive' reset event triggers a reset sequence
D_LVD27	**Disable 2.7V low-voltage detected** 0 A 2.7V low-voltage detected event triggers a reset sequence
D_SWT	**Disable software watchdog timer** 0 A software watchdog timer event triggers a reset sequence
D_LVD12_PD1	**Disable 1.2V low-voltage detected (power domain #1)** 0 A 1.2V low-voltage detected (power domain #1) event triggers a reset sequence
D_LVD12_PD0	**Disable 1.2V low-voltage detected (power domain #0)** 0 A 1.2V low-voltage detected (power domain #0) event triggers a reset sequence

9.3.1.5 Functional Event Alternate Request Register (RGM_FEAR)

Address 0xC3FE_4010 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R		AR_ST_NCF	AR_ST_CF			AR_Z4CORE			AR_LVD45	AR_CMU_FHL	AR_CMU_OLR	AR_FMPLL			AR_Z0CORE	AR_JTAG
W																
POR	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 96. Functional Event Alternate Request Register (RGM_FEAR)

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. Some fields are read-only, and their POR value and corresponding behavior cannot be changed. 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 99. Functional Event Alternate Request Register (RGM_FEAR) field descriptions

Field	Description
AR_ST_NCF	**Alternate Request for self-test non-critical fault** 1 Generate an interrupt request on a self-test non-critical fault event if the reset is disabled
AR_ST_CF	**Alternate Request for self-test critical fault** 0 Generate a SAFE mode request on a self-test critical fault event if the reset is disabled
AR_Z4CORE	**Alternate Request for e200z4d core reset** 0 Generate a SAFE mode request on a e200z4d core reset event if the reset is disabled 1 Generate an interrupt request on a e200z4d core reset event if the reset is disabled
AR_LVD45	**Alternate Request for 4.5V low-voltage detected** 0 Generate a SAFE mode request on a 4.5V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 4.5V low-voltage detected event if the reset is disabled

Table 99. Functional Event Alternate Request Register (RGM_FEAR) field descriptions

Field	Description
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_Z0CORE	**Alternate Request for e200z0h debug control core reset** 0 Generate a SAFE mode request on a e200z0h debug control core reset event if the reset is disabled 1 Generate an interrupt request on a e200z0h 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

9.3.1.6 Functional Event Short Sequence Register (RGM_FESS)

Address 0xC3FE_4018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS_EXR			SS_ST_DONE		SS_Z4CORE		SS_FLASH	SS_LVD45	SS_CMU_FHL	SS_CMU_OLR	SS_FMPLL	SS_CHKSTOP	SS_SOFT_FUNC	SS_Z0CORE	SS_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 97. Functional Event Short Sequence Register (RGM_FESS)

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.

Some fields are read-only, and their POR value and corresponding behavior cannot be changed. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 100. Functional Event Short Sequence Register (RGM_FESS) field descriptions

Field	Description
SS_EXR	**Short Sequence for External Reset** 0 The reset sequence triggered by an external reset event will start from PHASE1
SS_ST_DONE	**Short Sequence for self-test completed** 0 The reset sequence triggered by a self-test completed event will start from PHASE1
SS_Z4CORE	**Short Sequence for e200z4d core reset** 0 The reset sequence triggered by a e200z4d core reset event will start from PHASE1 1 The reset sequence triggered by a e200z4d core 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
SS_LVD45	**Short Sequence for 4.5V low-voltage detected** 0 The reset sequence triggered by a 4.5V low-voltage detected event will start from PHASE1 1 The reset sequence triggered by a 4.5V low-voltage detected event will start from PHASE3, skipping PHASE1 and PHASE2
SS_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
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_FUNC	**Short Sequence for software 'functional' reset** 0 The reset sequence triggered by a software 'functional' reset event will start from PHASE1 1 The reset sequence triggered by a software 'functional' reset event will start from PHASE3, skipping PHASE1 and PHASE2

Table 100. Functional Event Short Sequence Register (RGM_FESS) field descriptions (continued)

Field	Description
SS_Z0CORE	**Short Sequence for e200z0h debug control core reset** 0 The reset sequence triggered by a e200z0h debug control core reset event will start from PHASE1 1 The reset sequence triggered by a e200z0h 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

9.3.1.7 STANDBY Reset Sequence Register (RGM_STDBY)

This register controls booting 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.

This register is reset on any enabled ‘destructive’ or ‘functional’ reset event.

The CPU not used for booting on STANDBY exit remains disabled until it is enabled via the same mechanism that is used to enable the non-booting CPU after a reset.

When the Z4 is selected to boot from backup RAM on STANDBY exit, VLE code must be placed at the boot location in the RAM. The VLE code at this location can then program other memory pages to be BookE if required.

Note: If the "BOOT_FROM_BKP_RAM" bit is set to enable STANDBY exit to RAM, the e200z0 will be the CPU used by default. To enable the e200Z4 as the active core when exiting STANDBY to RAM, the SB_CPU bit must be set

Address 0xC3FE_401A Access: User read, Supervisor read/write, Test read/write

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

Figure 98. STANDBY Reset Sequence Register (RGM_STDBY)

Table 101. STANDBY Reset Sequence Register (RGM_STDBY) field descriptions

Field	Description
SB_CPU	STANBY0 Boot CPU — This bit selects which CPU is to be used for booting on STANDBY exit if the BOOT_FROM_BKP_RAM is set. If BOOT_FROM_BKP_RAM is cleared, the CPU used for booting after reset is used for booting on STANDBY exit. 0 Boot with the Z0 core on STANDBY exit 1 Boot with the Z4 core on STANDBY exit
BOOT_ FROM_ BKP_RAM	**Boot from Backup RAM indicator** — This bit indicates whether the system will boot from backup RAM or flash out of STANDBY exit. 0 Boot from flash on STANDBY exit 1 Boot from backup RAM on STANDBY exit (when using the Z4 from RAM in STANDBY ensure that VLE code is used, as described in this section). By default, the e200z0 will be the core that will boot on STANDBY exit when "Boot From RAM" is enabled. To change this to the e200z4, set the SB_CPU bit.

9.3.1.8 Functional Bidirectional Reset Enable Register (RGM_FBRE)

Address 0xC3FE_401C Access: User read, Supervisor read/write, Test read/write

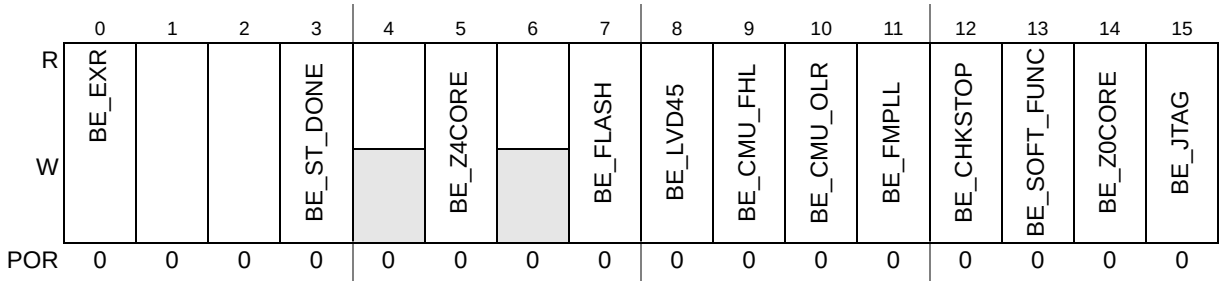


Figure 99. Functional Bidirectional Reset Enable Register (RGM_FBRE)

This register enables the generation of an external reset on functional reset. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.reset

Table 102. 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_ST_DONE	**Bidirectional Reset Enable for self-test completed** 0 RESET is asserted on a self-test completed event if the reset is enabled 1 RESET is not asserted on a self-test completed event
BE_Z4CORE	**Bidirectional Reset Enable for e200z4d core reset** 0 RESET is asserted on a e200z4d core reset event if the reset is enabled 1 RESET is not asserted on a e200z4d core 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

Table 102. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_LVD45	**Bidirectional Reset Enable for 4.5V low-voltage detected** 0 RESET is asserted on a 4.5V low-voltage detected event if the reset is enabled 1 RESET is not asserted on a 4.5V 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
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_FUNC	**Bidirectional Reset Enable for software 'functional' reset** 0 RESET is asserted on a software 'functional' reset event if the reset is enabled 1 RESET is not asserted on a software 'functional' reset event
BE_Z0CORE	**Bidirectional Reset Enable for e200z0h debug control core reset** 0 RESET is asserted on a e200z0h debug control core reset event if the reset is enabled 1 RESET is not asserted on a e200z0h 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 chip are reset based on the reset source event. This is summarized in *Table 103*.

Table 103. MC_RGM Reset implications

Source	What Gets Reset	External Reset Assertion(1)	Boot Mode Capture
power-on reset	all	yes	yes
‘destructive’ resets	all except some clock/reset management	yes	yes
external reset	all except some clock/reset management and debug	programmable(2)	yes

Table 103. MC_RGM Reset implications (continued)

Source	What Gets Reset	External Reset Assertion(1)	Boot Mode Capture
'functional' resets	all except some clock/reset management and debug	programmable(2)	programmable (3)
shortened 'functional' resets(4)	flip-flops except some clock/reset management	programmable(2)	programmable (3)

1. 'external reset assertion' means that the RESET pin is asserted by the MC_RGM until the end of reset PHASE3
2. the assertion of the external reset is controlled via the RGM_FBRE register
3. the boot mode is captured if the external reset is asserted
4. the short sequence is enabled via the RGM_FESS register

Note: JTAG logic has its own independent reset control and is not controlled by the MC_RGM in any way.

The reset sequence is comprised of five phases managed by a state machine, which ensures that all phases are correctly processed through waiting for a minimum duration and until all processes that need to occur during that phase have been completed before proceeding to the next phase.

The state machine used to produce the reset sequence is shown in *Figure 100*.

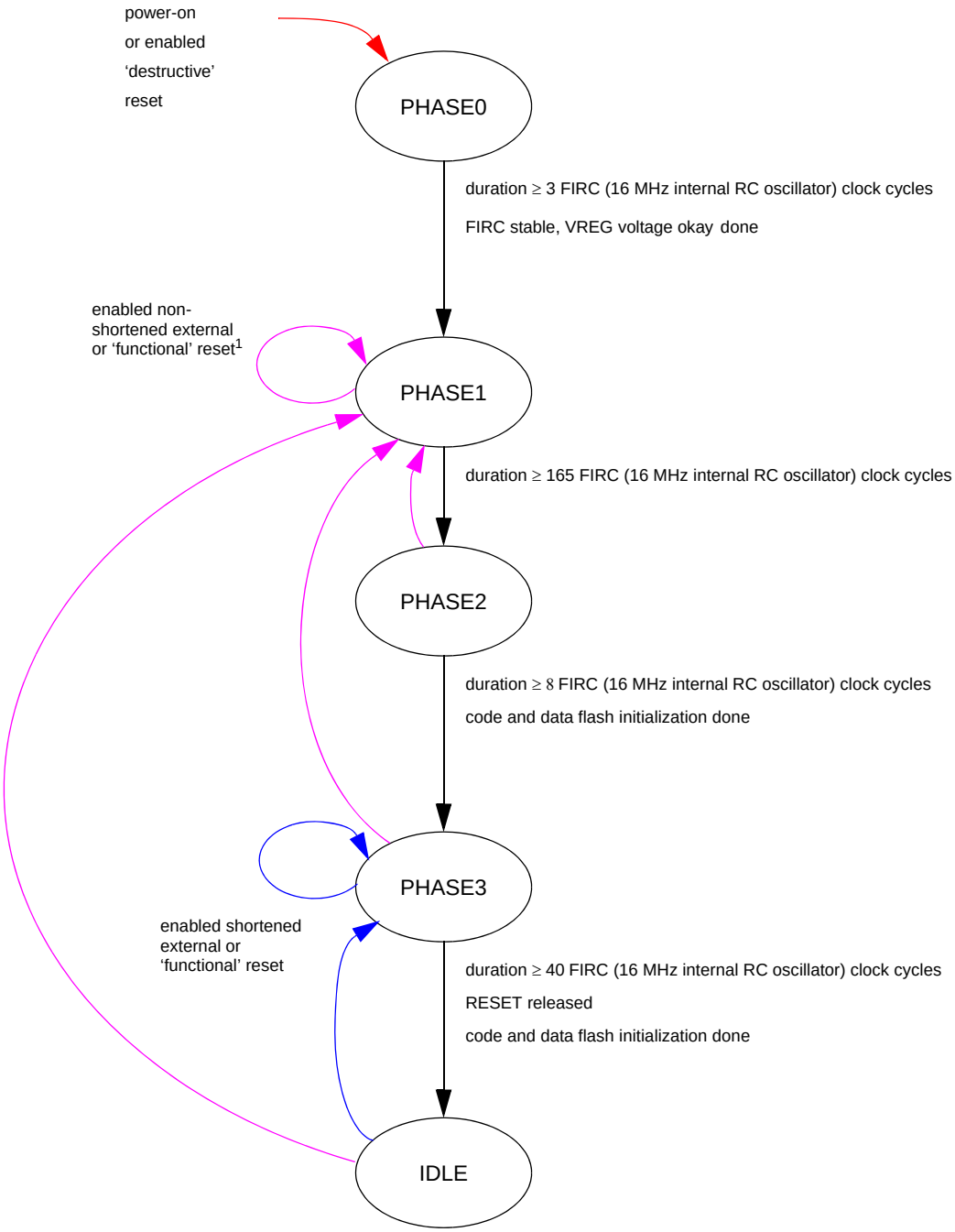


Figure 100. MC_RGM State Machine

9.4.1.1 PHASE0 phase

This phase is entered immediately from any phase on a power-on or enabled ‘destructive’ reset event. The reset state machine exits PHASE0 and enters PHASE1 on verification of the following:

- all enabled ‘destructive’ resets have been processed
- all processes that need to be done in PHASE0 are completed
  - FIRC stable, VREG voltage okay
- a minimum of 3 FIRC (16 MHz internal RC oscillator) clock cycles have elapsed since power-up completion and the last enabled ‘destructive’ reset event

9.4.1.2 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 165 FIRC (16 MHz internal RC oscillator) clock cycles have elapsed since the last enabled external or non-shortened ‘functional’ reset event

9.4.1.3 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 FIRC (16 MHz internal RC oscillator) clock cycles have elapsed since entering PHASE2

9.4.1.4 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 FIRC (16 MHz internal RC oscillator) clock cycles have elapsed since the last enabled, shortened ‘functional’ reset event

9.4.1.5 IDLE phase

This is the final phase and is entered on exit from PHASE3. When this phase is reached, the MC_RGM releases control of the chip to the core and waits for new reset events that can trigger a reset sequence.

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 chip’s low-voltage detector threshold ensures that, when 1.2V 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 chip.

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 some reset source events to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the RGM_FERD and RGM_FEAR registers as shown in *Table 104*.

Table 104. MC_RGM alternate event selection

RGM_FERD Bit Value	RGM_FEAR Bit Value	Generated Event
0	X	reset
1	0	SAFE mode request
1	1	interrupt request

The alternate event is cleared by deasserting the source of the request (that is, at the reset source that caused the alternate request) and also clearing the appropriate RGM_FES status bit.

Note: Alternate requests (SAFE mode as well as interrupt requests) are generated regardless of whether the system clock is running.

*Note: If a masked 'functional' reset event which is configured to generate a SAFE mode/interrupt request occurs during PHASE1, it **is ignor**ed, and the MC_RGM will not send any safe mode/interrupt request to the MC_ME.*

9.4.7 Boot mode capturing

The MC_RGM samples PA[9:8] whenever RESET is asserted until five FIRC (16 MHz internal RC oscillator) clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after RESET has been deasserted for subsequent boots after reset sequences during which RESET is not asserted.

Note: In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that RESET is asserted.

Note: RESET can be asserted as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See Table 103 for details.)

10 Power Control Unit (MC_PCU)

10.1 Introduction

10.1.1 Overview

The power control unit (MC_PCU) is used to control the overall microcontroller power consumption. There are 4 power domains controlled by the MC_PCU which physically disconnects power to the domains when the specific operating mode allows this.

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.

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 101 depicts the MC_PCU block diagram.

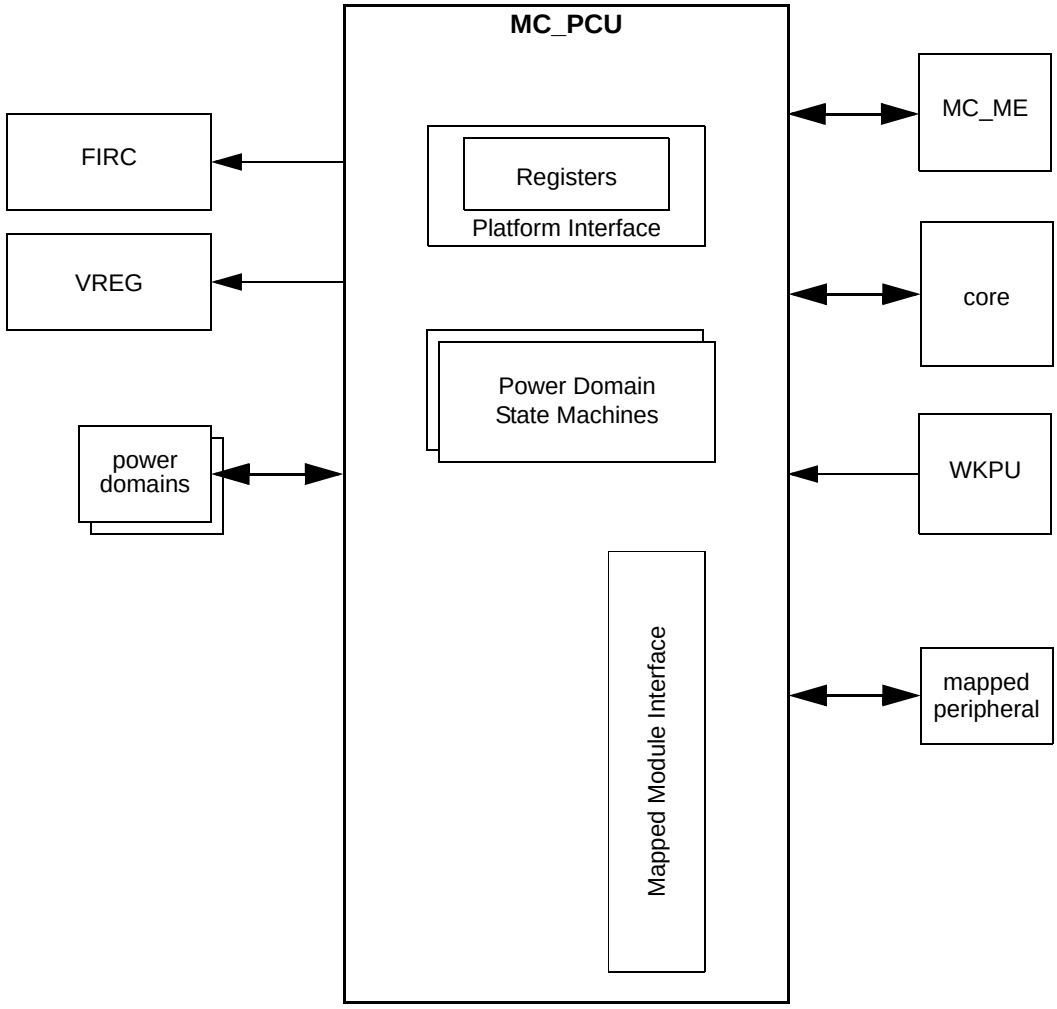


Figure 101. MC_PCU block diagram

10.1.2 Features

The MC_PCU includes the following features:

- support for 4 power domains
- support for chip 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

10.2 External Signal Description

The MC_PCU has no connections to any external pins.

10.3 Memory Map and Register Definition

10.3.1 Memory Map

Table 105. MC_PCU Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_8000	PCU_PCONF0	Power Domain #0 Configuration	word	read	read	read	*on page 305*
0xC3FE_8004	PCU_PCONF1	Power Domain #1 Configuration	word	read	read	read	*on page 306*
0xC3FE_8008	PCU_PCONF2	Power Domain #2 Configuration	word	read	read/write	read/write	*on page 306*
0xC3FE_800C	PCU_PCONF3	Power Domain #3 Configuration	word	read	read/write	read/write	*on page 306*
0xC3FE_8040	PCU_PSTAT	Power Domain Status Register	word	read	read	read	*on page 308*

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 106. MC_PCU Memory Map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_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 … 0xC3FE_800C	PCU_PCONF2…3	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	STBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																

Table 106. MC_PCU Memory Map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_8010 … 0xC3FE_803C	reserved																	
0xC3FE_8040	PCU_PSTAT	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R													PD3	PD2	PD1	PD0
		W																
0x044 … 0x07C	reserved																	
0xC3FE_8080 … 0xC3FE_80FC	VREG registers (For details, see the VREG Register Description section of the Reference Manual)																	
0xC3FE_8100 … 0xC3FE_BFFC	reserved																	

10.3.2 Register descriptions

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

10.3.2.1 Power Domain #0 Configuration Register (PCU_PCONF0)

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

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	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

Figure 102. Power Domain #0 Configuration Register (PCU_PCONF0)

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

Table 107. 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

Table 107. Power Domain Configuration register field descriptions (continued)

Field	Description
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

10.3.2.2 Power Domain #1 Configuration Register (PCU_PCONF1)

Address 0xC3FE_8004 Access: User read, Supervisor read, Test read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	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

Figure 103. Power Domain #1 Configuration Register (PCU_PCONF1)

This register defines for power domain #1 whether it is on or off in each chip mode. The bit field description is the same as in *Table 107*. 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 refer to *Section 10.4.4.2: STANDBY Mode transition*.

10.3.2.3 Power Domain #2…3 Configuration Registers (PCU_PCONF2…3)

These registers define for each power domain #2 through #3 whether it is on or off in each chip mode. The bit field description is the same as in *Table 107*.

Power domains #2 and #3 contain the backup RAM extensions and can only be powered down during STANDBY mode.Therefore, writing a '0' to the respective bits for the other modes will not power down those domains.

There are 4 possible configurations for the amount of RAM active in STANDBY mode as shown *Table 108* and *Table 109*. The 40 KB option results in a non-contiguous address space.

Table 108. RAM configurations in modes

Mode(s)	8 KB 0x4000_0000 0x4000_1FFF	56 KB 0x4000_2000 0x4000_FFFF	32 KB 0x4001_0000 0x4001_7FFF	32 KB 0x4001_8000 0x4001_FFFF	128 KB 0x4002_0000 0x4003_FFFF
RUN, STOP, HALT	On(1)	On(1)	On(1)	On(1)	On(1)
STANDBY	On	PCONF[2]	PCONF[3]	Off	Off

1. If the MVRON bit is cleared in the ME_<mode>_MC register for STOP or HALT modes, the SRAM is placed into a low power state where it cannot be accessed. The SRAM contents are retained and will be available again after exiting the low power mode.

Table 109. PCONF settings for RAM in STANDBY

STANDBY RAM (KB)	PCONF[2]	PCONF[3]
8	0	0
64 (8 + 56)	1	0
96 (8 + 56 + 32)	1	1
40 (8 + 32)(1)	0	1

1. Results in non-contiguous address space

Address 0xC3FE_8008 - 0xC3FE_800C

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

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY	0	0	STOP(1)	0	HALT(1)	RUN3(1)	RUN2(1)	RUN1(1)	RUN0(1)	DRUN(1)	SAFE(1)	TEST(1)	RST
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1

1 The microcontroller allows you to write to these fields in all modes, but you must not do so except in STANDBY mode. Writing to these fields in any other mode is an illegal operation that results in undetermined results.

Figure 104. Power Domain #2…3 Configuration Registers (PCU_PCONF2…3)

10.3.2.4 Power Domain Status Register (PCU_PSTAT)

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

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R													PD3	PD2	PD1	PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 105. Power Domain Status Register (PCU_PSTAT)

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

Table 110. 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 chip 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 chip 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.

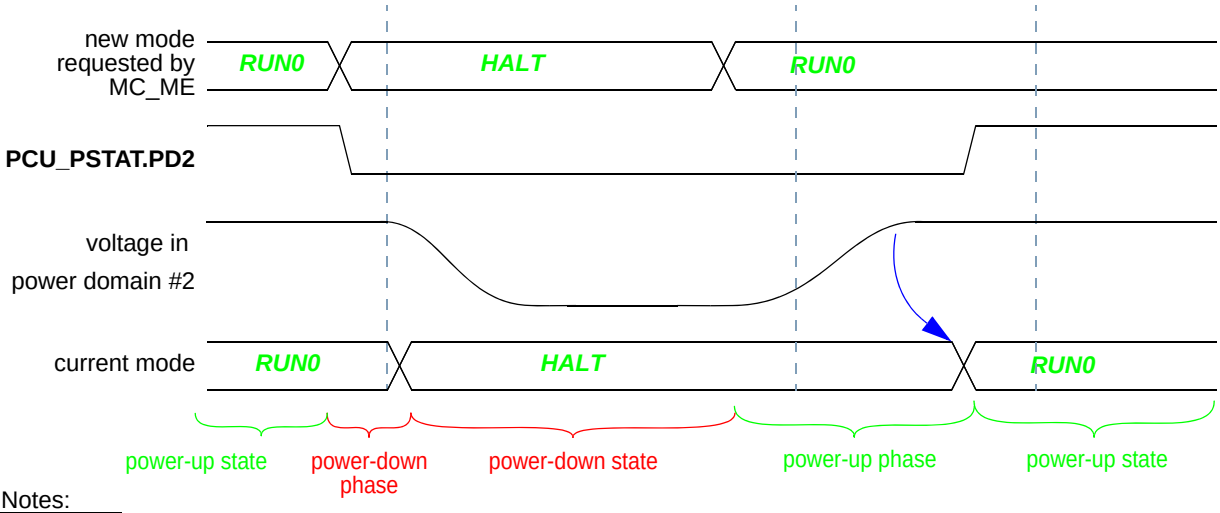
10.4.4.1 DRUN, SAFE, TEST, RUN0…3, HALT, and STOP mode transition

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 106 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).



Notes:

Not drawn to scale; PCONF2.RUN0 = 1; PCONF2.HALT = 0

Figure 106. 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.

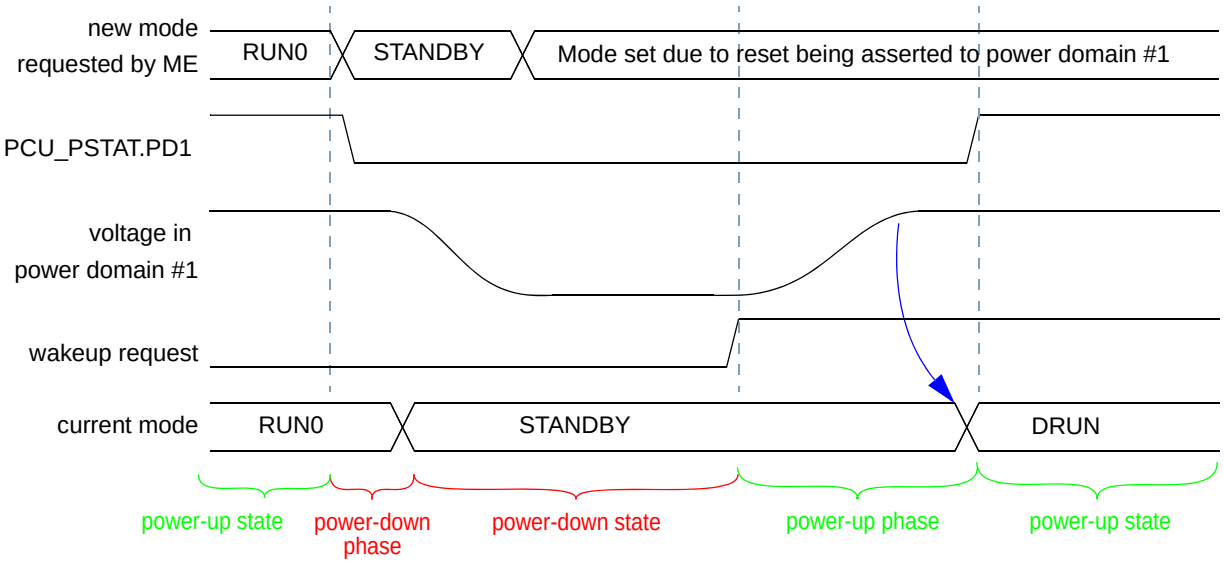
10.4.4.2 STANDBY Mode transition

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 shows an example for a mode transition from RUN0 to STANDBY to DRUN. All power domains which have PCU_PCONFn.STBY 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 107. 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).

Prior to standby entry, PCTL for WKPU should be disabled.

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.

10.4.4.3 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…3 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 chip 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 V–5 V ± 10%) and digital/regulated output supply is (1.2 V ± 10%). The voltage regulator used in SPC564Bxx and SPC56ECxx comprises two regulators.

- High power regulator (HPREG)
- Low power regulator (LPREG)

The HPREG regulator is switched off during the STANDBY mode to save consumption from the regulator itself. 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 switched-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. Care should also be taken to limit the serial inductance of the board.

Figure 110 shows the power domain organization of SPC564Bxx and SPC56ECxx. The digital design is partitioned into two main domains:

- PD1 — Switchable domain (powered off in standby mode)
- PD0 — Always powered on

Note: SPC564Bxx and SPC56ECxx also contains PD2 and PD3 domains for controlling additional SRAM and STANDBY. For details, refer to Section 10.3.2: Register descriptions.

These domains are connected through low leakage power switches inside the design. These switches open during STANDBY mode, thus providing power gating to PD1. For both domains, there is a low voltage detector for the 1.2 V output voltage. Additionally, there are two low voltage detectors for the VDD_HV_A, one at the 3.3V level and one at the 5V level. VDD_HV_A is also use to power the on-chip regulators (HPREG and LPREG).

11.1.1 High power regulator (HPREG)

The HPREG converts the 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.

In STOP and HALT modes, a request can be made to disable the regulator by clearing the MVRON bit in the ME_<mode>_MC register. The regulator will be disabled once system power consumption falls to a sufficient level. If the main regulator is disabled, the SRAM is put into a low power state and cannot be read or written, although the contents are retained.

11.1.2 Low power regulator (LPREG)

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

11.1.3 LVDs and POR

There are three types of LVD available:

1. LVD_MAIN for the 3.3 V–5 V input supply
2. LVD_MAIN5 for the 3.3 V–5 V input supply
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 PD1 domain voltage notifying that the 1.2 V output is stable. The other LVD_DIG is placed in the standby domain (PD0) 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 reach the desired range. Power-down controls are provided for LVDs. When LVDs are power-down, their outputs are de-asserted. No trimming capability is provided on LVD_MAIN and LVD_MAIN5. The trimming is done through flash and happens in phase3 of the reset sequencing.

POR is required to initialize the device during supply rise. POR works when the main supply (VDD_HV_A) is ramping up. 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. To assert POR again, Vdd needs to ramp down to V_{PORUP} min levels.

11.1.4 VREG digital interface

The voltage regulator digital interface provides the temporization delay at initial power-up and protocol required during entry and exit from low-power modes (STOP/STANDBY). A signal indicating that Low Power domain is powered is used at power-up to release reset to temporization counter. At exit from low-power modes (STOP with Vreg-off / STANDBY), 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, an end-of-count signal is released, it is gated with another signal indicating the main domain voltage correct to release the VREG_OK signal. This is used by MC_RGM to release the reset to the device and enter to the next phase, i.e. phase1. It also manages trimming of HPREG, LPREG during RUN, STOP, or STANDBY mode. It also manages transition from run to low power mode and vice-versa.

11.1.4.1 Features

- Temporization counter of 18 µs to get HPREG up and stable during:
  - initial power up
  - STOP (with vreg-off) exit
  - STANDBY mode exit
- Transition between different modes:
  - RUN to STOP with vreg-off
  - RUN to STANDBY mode
  - STOP with Vreg-off to RUN mode
  - STANDBY to RUN mode
- Control register to mask 5 V LVD when the regulator input supply is 3.3 V
- Control register to switch-off the PORPU circuitry of voltage regulator when HPREG is powered down (either during STOP or STANDBY)

11.1.5 Memory map

The registers in this section are mapped to the MC_PCU address space as described in *Chapter 10: Power Control Unit (MC_PCU)*.

Table 111. Memory map

Base address: 0xC3FE_8080		
Address offset	**Register**	**Location**
0x0	Voltage Regulator Control Register (VREG_CTL)	*on page 314*
0x4	Voltage Regulator Power Down mode VREG_PDMODE	*on page 315*

11.1.6 Register description

11.1.6.1 Voltage Regulator Control Register (VREG_CTL)

Voltage Regulator Control Register (VREG_CTL) controls the masking of the 5 V LVD. It is used for disabling/enabling the 5 V LVD.

Offset: 0x0 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

Figure 108. Voltage Regulator Control Register (VREG_CTL)

Table 112. 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 ‘0’ 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.

11.1.6.2 Voltage Regulator (VREG_PDMODE)

This register is implemented to support programming of PORPU circuitry in voltage regulator of current when HPREG is powered down during STANDBY mode. For SRAM data retention, minimum power supply varies according to memory operating modes.

Offset: 0x0 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	PORPU
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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	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

Figure 109. Voltage Regulator Register (VREG_PDMODE)

Table 113. VREG_PDMODE field descriptions

Field	Description
PORPU	This bit is used to control the i/p signal PORPU of voltage regulator during STOP/STANDBY mode. 1 : POR circuitry is enabled 0 : POR circuitry is disabled.

11.2 Power supply strategy

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

The device provides the following dedicated supply domains at package level:

- HV: High voltage external power supply for voltage regulator module. This must be provided externally through VDD_HV_A power pins.
- LV: Low voltage internal power supply for core, FMPLL and Flash digital logic. This is generated by the on-chip VREG with an external ballast (NPN device). It is further split into four main domains to ensure noise isolation between critical LV modules within the device:
  - LV_COR: Low voltage supply for the core. It is also used to provide supply for FMPLL through double bonding.
  - LV_CFLA0/CFLA1: Low voltage supply for the two code Flash modules.
  - LV_DFLA: Low voltage supply for data flash module.
  - LV_PLL: Low voltage supply for FMPLL.

For further details, refer to Power management electrical characteristics section of Data Sheet.

11.3 Power domain organization

This device employs four primary power domains, namely PD0, PD1, PD2 and PD3.Power domain organization and connections to the internal regulator are depicted in *Figure 110*.

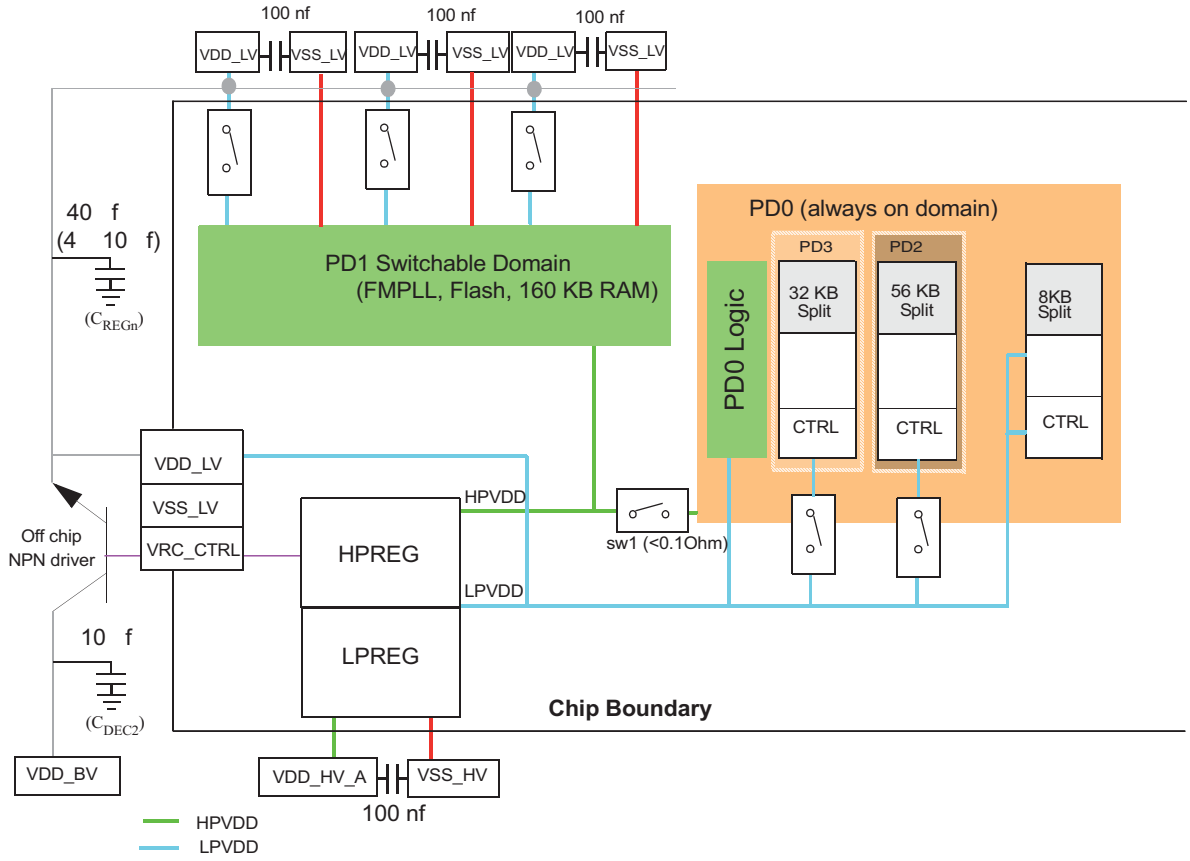


Figure 110. Power domain organization

12 Wakeup Unit (WKPU)

12.1 Overview

The Wakeup Unit supports 2 internal sources and 30 external sources that can generate interrupts or wakeup events, of which 2 can cause non-maskable interrupt requests or wakeup source. *Figure 111* is a block diagram of the Wakeup Unit and its interfaces to other system components.

The wakeup lines are mapped on the interrupt vectors as shown in *Table 114*. 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 114. 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		
								176-pin QFP	208-pin QFP	256-pin BGA
WKPU0	API	n/a(3)	—	WakeUp_IRQ_0	46	EIF0	31	✓(3)	✓(3)	✓(3)
WKPU1	RTC	n/a(3)	—			EIF1	30	✓(3)	✓(3)	✓(3)
WKPU2	PA1	PCR1	NMI[0], CAN3RX			EIF2	29	✓	✓	✓
WKPU3	PA2	PCR2	NMI[1]			EIF3	28	✓	✓	✓
WKPU4	PB1	PCR17	CAN0RX, LIN0RX			EIF4	27	✓	✓	✓
WKPU5	PC11	PCR43	CAN1RX, CAN4RX			EIF5	26	✓	✓	✓
WKPU6	PE0	PCR64	CAN5RX			EIF6	25	✓	✓	✓
WKPU7	PE9	PCR73	CAN2RX, CAN3RX			EIF7	24	✓	✓	✓
WKPU8	PB10	PCR26	—	WakeUp_IRQ_1	47	EIF8	23	✓	✓	✓
WKPU9	PA4	PCR4	LIN5RX			EIF9	22	✓	✓	✓
WKPU10	PA15	PCR15	—			EIF10	21	✓	✓	✓
WKPU11	PB3	PCR19	LIN0RX			EIF11	20	✓	✓	✓
WKPU12	PC7	PCR39	LIN1RX			EIF12	19	✓	✓	✓
WKPU13	PC9	PCR41	LIN2RX			EIF13	18	✓	✓	✓
WKPU14	PE11	PCR75	LIN3RX			EIF14	17	✓	✓	✓
WKPU15	PF11	PCR91	LIN4RX			EIF15	16	✓	✓	✓

Table 114. 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		
								176-pin QFP	208-pin QFP	256-pin BGA
WKPU16	PF13	PCR93	LIN5RX	WakeUp_IRQ_2	48	EIF16	15	✓	✓	✓
WKPU17	PG3	PCR99	—			EIF17	14	✓	✓	✓
WKPU18	PG5	PCR101	—			EIF18	13	✓	✓	✓
WKPU19	PA0	PCR0	CAN1RX			EIF19	12	✓	✓	✓
WKPU20	PG7	PCR103	LIN6RX			EIF20	11	✓	✓	✓
WKPU21	PG9	PCR105	LIN7RX			EIF21	10	✓	✓	✓
WKPU22	PF9	PCR89	CAN2RX, CAN3RX			EIF22	9	✓	✓	✓
WKPU23	PI3	PCR131	LIN9RX			EIF23	8	✓	✓	✓
WKPU24	PI1	PCR129	LIN8RX	WakeUp_IRQ_3	49	EIF24	7	✓	✓	✓
WKPU25	PB8	PCR24	—			EIF25	6	✓	✓	✓
WKPU26	PB9	PCR25	—			EIF26	5	✓	✓	✓
WKPU27	PD0	PCR48	—			EIF27	4	✓	✓	✓
WKPU28	PD1	PCR49	—			EIF28	3	✓	✓	✓
WKPU29	PE[3]	PCR67	FR_A_RX			EIF29	2	✓	✓	✓
WKPU30	PE[5]	PCR69	FR_B_RX			EIF30	1	✓	✓	✓
WKPU31	PJ[13]	PCR157	CAN1RX, CAN4RX			EIF31	0	x[4]	✓	✓

1. This column does not contain an exhaustive list of functions on that pin. Rather, it includes peripheral communication functions (such as CAN and LINFlexD 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.

Note: There is no dedicated wake up pin for the replicated mux functions on the 208 package. Therefore, CANRX_2, CANRX_3, CANRX_5, LINRX_2, LINRX_3 and LINRX_8 do not have wakeup assigned in every position that they occur.

Note: 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. Details of non-bonded wake-ups for the smaller packages given below:

176-pin LQFP – PJ[13]

Note: In HALT mode and in STOP mode where the system clock is still enabled, an external interrupt (EIRQ) or any peripheral interrupt can be used to wake the device up.

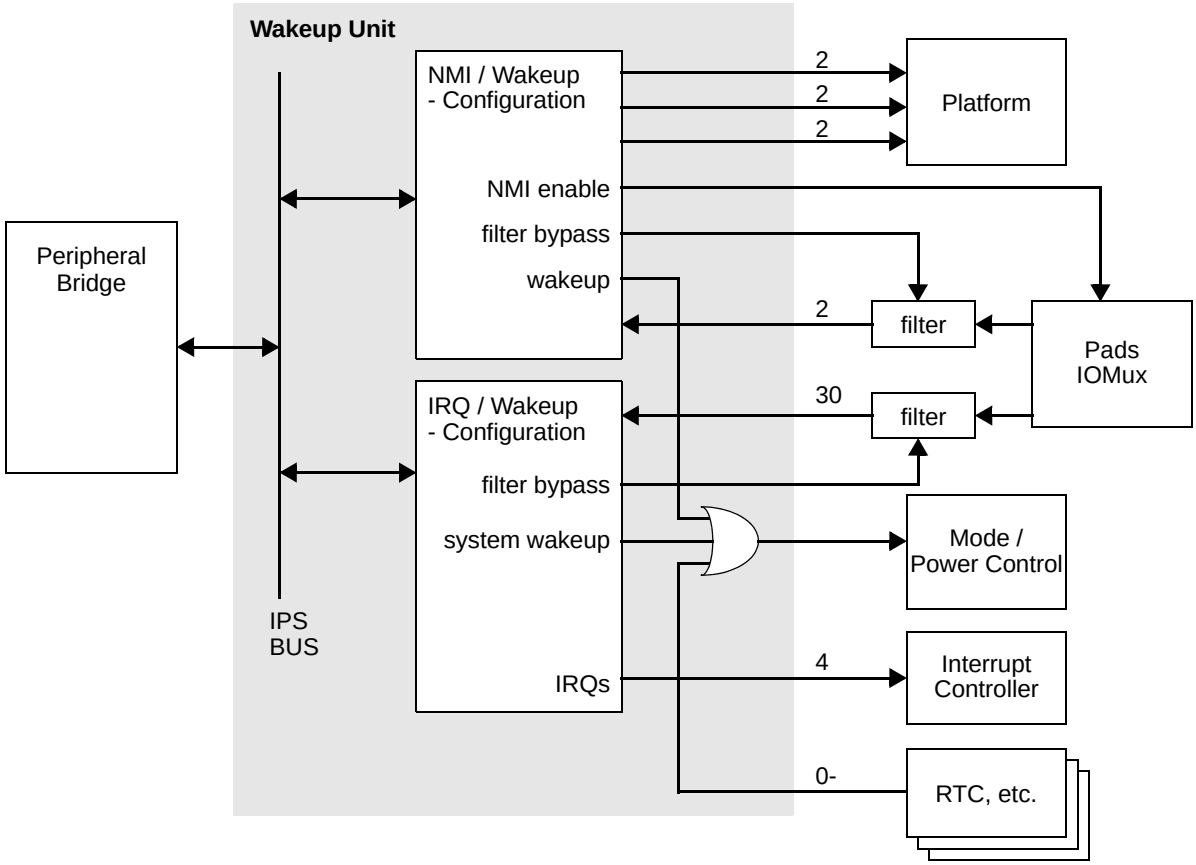


Figure 111. Wakeup unit block diagram

12.2 Features

The Wakeup Unit supports these distinctive features:

- Non-maskable interrupt support with
  - 2 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
  - 30 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 Memory map and register description

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

12.3.1 Memory map

Table 115 gives an overview on the WKPU registers implemented.

Table 115. WKPU memory map

Base address: 0xC3F9_4000		
Address offset	**Register**	**Location**
0x0000	NMI Status Flag Register (NSR)	*on page 322*
0x0004 – 0x0007	Reserved	—
0x0008	NMI Configuration Register (NCR)	*on page 323*
0x000C – 0x0013	Reserved	—
0x0014	Wakeup/Interrupt Status Flag Register (WISR)(1)	*on page 325*
0x0018	Interrupt Request Enable Register (IRER)	*on page 325*
0x001C	Wakeup Request Enable Register (WRER)(1)	*on page 326*
0x0020 – 0x0027	Reserved	—
0x0028	Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)(1)	*on page 326*

Table 115. WKPU memory map (continued)

Base address: 0xC3F9_4000		
Address offset	**Register**	**Location**
0x002C	Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)(1)	*on page 327*
0x0030	Wakeup/Interrupt Filter Enable Register (WIFER)	*on page 327*
0x0034	Wakeup/Interrupt Pullup Enable Register (WIPUER)	*on page 328*
0x0038 – 0x03FFF	Reserved	—

1. Applies to both external and internal wakeup sources.

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.

12.3.2 Register description

This section describes in address order all the Wakeup Unit registers. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of register is MSB = 0, however the numbering of internal field is LSB = 0, for example EIF[5] = WISR[26].

Each bit of Wakeup Unit registers can be mapped to a wake-up port.

Figure 112 shows wake-up ports mapped to register bits.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Port	PJ13	PE5	PE3	PD1	PD0	PB9	PB8	PI1	PI3	PF9	PG9	PG7	PA0	PG5	PG3	PF13
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Port	PF11	PE11	PC9	PC7	PB3	PA15	PA4	PB10	PE9	PE0	PC11	PB1	PA2	PA1	RTC	API

Figure 112. Wake-up port mapping

12.3.2.1 NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Offset: 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NIF[0]	NOVF[0]	0	0	0	0	0	0	NIF[1]	NOVF[1]	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

Figure 113. NMI Status Flag Register (NSR)

Table 116. NSR field descriptions

Field	Description
NIF[0]	NMI 0 Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NREE or NFEE set), NIF[0] causes an interrupt request. 1 An event as defined by NREE and NFEE has occurred 0 No event has occurred on the pad
NOVF[0]	NMI 0 Overrun Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. It will be a copy of the current NIF[0] value whenever an NMI event occurs, thereby indicating to the software that an NMI 0 occurred while the last one was not yet serviced. If enabled (NREE or NFEE set), NOVF causes an interrupt request. 1 An overrun has occurred on NMI 0 input 0 No overrun has occurred on NMI 0 input
NIF[1]	NMI 1 Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NREE or NFEE set), NIF [1] causes an interrupt request. 1 An event as defined by NREE and NFEE has occurred 0 No event has occurred on the pad
NOVF[1]	NMI 1 Overrun Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. It will be a copy of the current NIF[1] 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 (NREE or NFEE set), NOVF causes an interrupt request. 1 An overrun has occurred on NMI 1 input 0 No overrun has occurred on NMI 1 input

12.3.2.2 NMI Configuration Register (NCR)

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

Offset: 0x0008 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	NLOCK1	NDSS1		NWRE 1	o	NREE1	NFEE1	NFE1
W																
Reset	0	1	1	0	0	0	0	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	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

Figure 114. NMI Configuration Register (NCR)

Table 117. NCR field descriptions

Field	Description
NLOCK0	NMI Configuration Lock Register for e200z0 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 for e200z0 00 Non-maskable interrupt 01 Critical interrupt 10 Machine check request 11 Reserved—no NMI, critical interrupt, or machine check request generated
NWRE0	NMI Wakeup Request Enable for e200z0 1 A set NIF bit or set NOVF bit causes a system wakeup request 0 System wakeup requests from the corresponding NIF 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 for e200z0 1 Rising-edge event is enabled 0 Rising-edge event is disabled
NFEE0	NMI Falling-edge Events Enable for e200z0 1 Falling-edge event is enabled 0 Falling-edge event is disabled
NFE0	NMI Filter Enable for e200z0 Enable analog glitch filter on the NMI pad input. 1 Filter is enabled 0 Filter is disabled
NLOCK1	NMI Configuration Lock Register for e200z4 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.
NDSS1	NMI Destination Source Select for e200z4 00 Non-maskable interrupt 01 Critical interrupt 10 Machine check request 11 Reserved—no NMI, critical interrupt, or machine check request generated

Table 117. NCR field descriptions (continued)

Field	Description
NWRE1	NMI Wakeup Request Enable for e200z4 1 A set NIF bit or set NOVF bit causes a system wakeup request 0 System wakeup requests from the corresponding NIF bit are disabled
NREE1	NMI Rising-edge Events Enable for e200z4 1 Rising-edge event is enabled 0 Rising-edge event is disabled
NFEE1	NMI Falling-edge Events Enable for e200z4 1 Falling-edge event is enabled 0 Falling-edge event is disabled
NFE1	NMI Filter Enable for e200z4 Enable analog glitch filter on the NMI pad input. 1 Filter is enabled 0 Filter is disabled

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

12.3.2.3 Wakeup/Interrupt Status Flag Register (WISR)

This register holds the wakeup/interrupt flags.

Offset: 0x0014 Access: User read/write (write 1 to clear)

	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	EIF[31:0]																															
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

Figure 115. Wakeup/Interrupt Status Flag Register (WISR)

Table 118. 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

12.3.2.4 Interrupt Request Enable Register (IRER)

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

Offset: 0x0018 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	EIRE[31:0]																															
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

Figure 116. Interrupt Request Enable Register (IRER)

Table 119. 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

12.3.2.5 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.

Offset: 0x001C 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	WRE[31:0]																															
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

Figure 117. Wakeup Request Enable Register (WRER)

Table 120. 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

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

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

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

Offset: 0x0028 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	IREE[31:0]																															
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

Figure 118. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)

Table 121. 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

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

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

Offset: 0x002C 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	IFEE[31:0]																															
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

Figure 119. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)

Table 122. 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

12.3.2.8 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.

Offset: 0x0030 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	IFE[31:0]																															
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

Figure 120. Wakeup/Interrupt Filter Enable Register (WIFER)

Table 123. 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

12.3.2.9 Wakeup/Interrupt Pullup Enable Register (WIPUER)

This register is used to enable a pull up on the corresponding interrupt pads to pull a wakeup/interrupt input to a value of '1'.

Offset: 0x0034 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	IPUE [31:0]																															
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

Figure 121. Wakeup/Interrupt Pullup Enable Register (WIPUER)

Table 124. WIPUER field descriptions

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

Note: A wakeup/interrupt pad configuration for a pull up through the WIPUER will be activated on the next STANDBY mode entry.

Related internal pull up resistor is controlled by WKPU during STANDBY, but it is overwritten by SIUL during RUN mode. Therefore any pull up configuration active during STANDBY will be lost after wakeup in RUN mode.

12.4 Functional description

12.4.1 General

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

12.4.2 Non-maskable interrupts

The System Integration Unit Lite supports up to two non-maskable interrupts.

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.

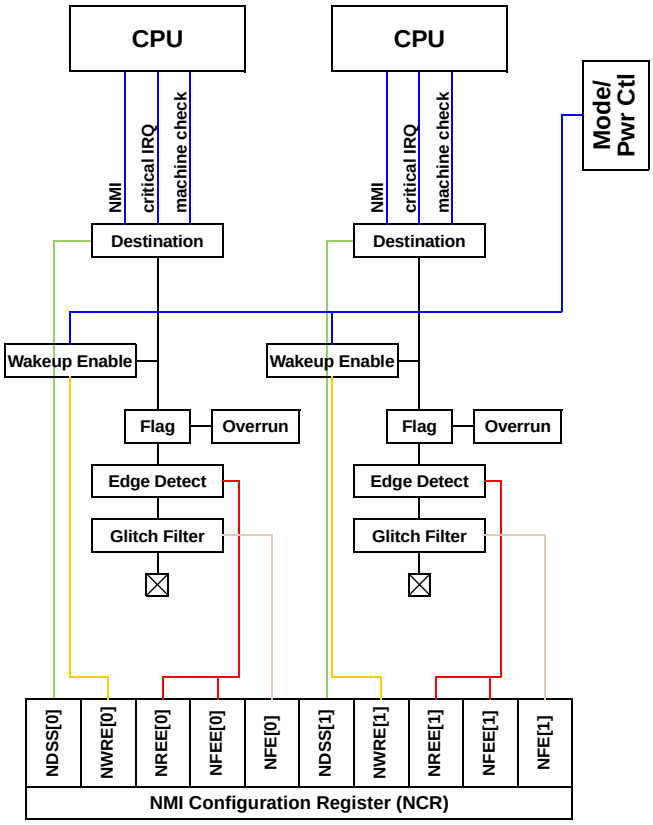


Figure 122. NMI pad diagram

12.4.2.1 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 114*). The pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled results in no interrupt being detected and should not be configured.

The active NMI edge is controlled by the user through the configuration of the NREE and NFEE bits.

Note: After reset, NREE and NFEE are set to '0', therefore the NMI functionality is disabled after reset and must be enabled explicitly by software.

Once the pad's NMI functionality has been enabled, the pad cannot be reconfigured in the IOMUX to override or disable the NMI.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS bits. See *Table 117* for details.

An NMI supports a status flag and an overrun flag which are located in the NSR register (see *Figure 113*). This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register. The status flag is set whenever an NMI event

is detected. The overrun flag is set whenever an NMI event is detected and the status flag is set (that is, has not yet been cleared).

Note: The overrun flag is cleared by writing a '1' to the appropriate overrun bit in the NSR register. If the status bit is cleared and the overrun bit is still set, the pending interrupt will not be cleared.

12.4.3 External wakeups/interrupts

The Wakeup Unit supports up to four 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.

Refer to *Figure 123* for an overview of the external interrupt implementation for the example of four interrupt vectors with up to eight external interrupt sources each.

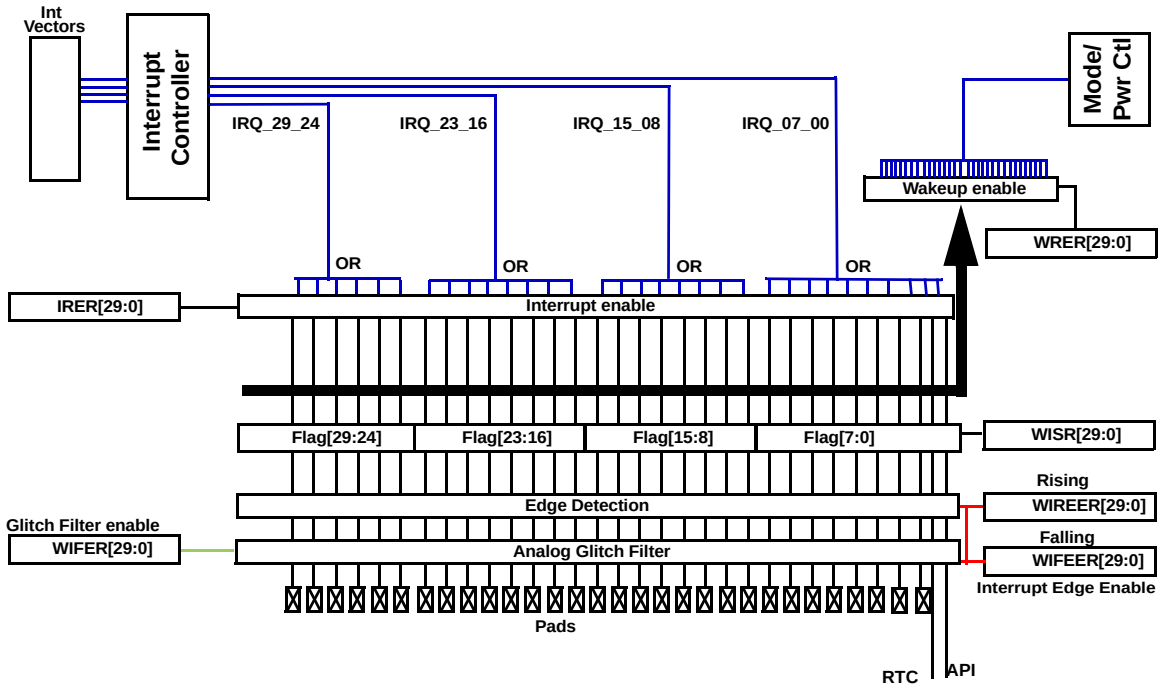


Figure 123. External Interrupt Pad Diagram

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.

12.4.3.1 External interrupt management

Each external interrupt can be enabled or disabled independently. This can be performed using a single rolled up register (*Figure 116*). 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). This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.

12.4.4 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.

12.4.4.1 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 115* for details).

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

13.1 Overview

The RTC 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). When the RTC interval is reached, the RTC will generate a wakeup request to the wakeup unit. The RTC can be configured to generate an interrupt on the RTC interval timeout.

The RTC also supports an autonomous periodic interrupt (API) function which can be used to generate a periodic event to the wakeup unit or an interrupt request.

13.2 Features

Features of the RTC/API include:

- 4 selectable counter clock sources
  - SIRC (128 kHz)
  - SXOSC (32 kHz)
  - FIRC (16 MHz)
  - FXOSC (4-40 MHz)
- Clock sources can optionally be prescaled by 512 or 32
- 32-bit counter
  - Supports times up to 1.5 months with 1 ms resolution
  - Runs in all modes of operation, including normal RESET
  - 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 when using 32 Khz external oscillator source with divide by 32 enabled
- 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 when using 32 Khz external oscillator source with divide by 32 enabled
  - 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

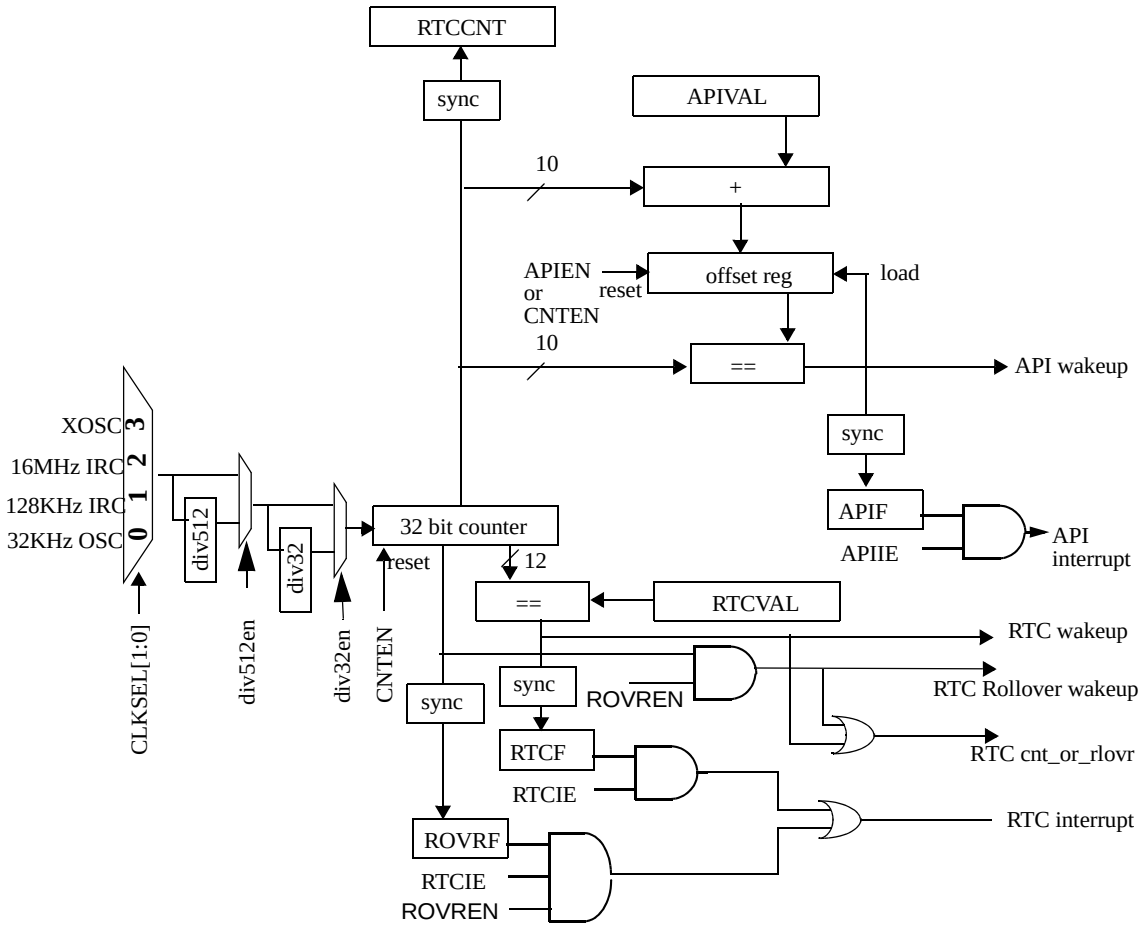


Figure 124. RTC/API block diagram

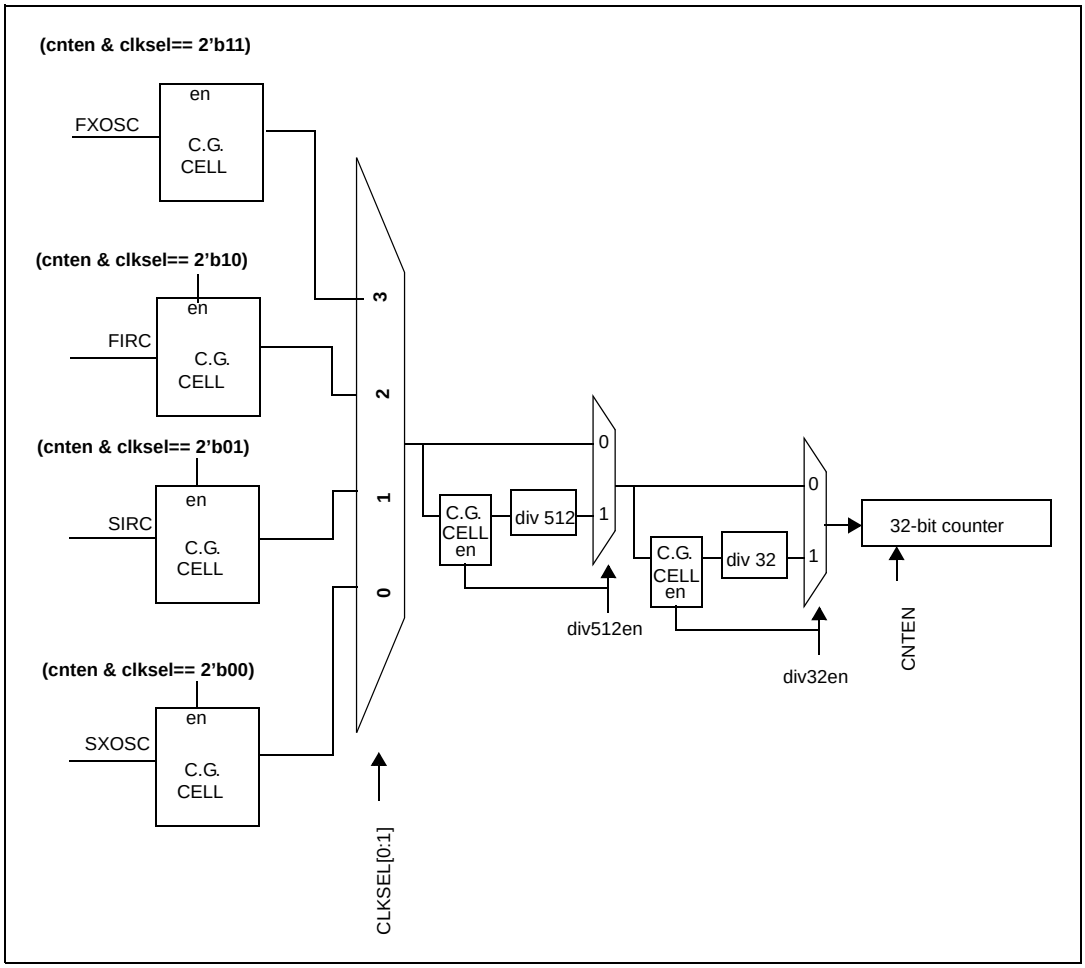


Figure 125. Clock gating for RTC clocks

13.3 Modes of operation

13.3.1 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. The RTC/API and associated interrupts are optionally enabled. In low power mode, the bus interface is disabled and no configuration changes are permitted. The RTC/API is enabled if enabled prior to entry into low power mode.

13.3.2 Debug mode

If RTCC[FRZEN] is set, the counter will stop on the last valid count when the device enters debug mode. On debug mode exit, the counter will resume counting from the frozen value. If RTCC[FRZEN] is clear, the counter will continue counting (and set flags accordingly) when the device is in debug mode.

13.4 Register descriptions

The registers listed in *Table 125* are described in the following sections.

Table 125. RTC/API register map

Base address: 0xC3FE_C000		
Address offset	**Register**	**Location**
0x0000	RTC Supervisor Control Register (RTCSUPV)	*on page 335*
0x0004	RTC Control Register (RTCC)	*on page 336*
0x0008	RTC Status Register (RTCS)	*on page 338*
0x000C	RTC Counter Register (RTCCNT)	*on page 339*

13.4.1 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.

Address: RTC_BASE + 0x0000 Access:

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

	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

Figure 126. RTC Supervisor Control Register (RTCSUPV)

Table 126. RTCSUPV register 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.

13.4.2 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

Address: RTC_BASE + 0x0004 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

Figure 127. RTC Control Register (RTCC)

Table 127. RTCC register field descriptions

Field	Description
CNTEN	Counter Enable The CNTEN bit enables the RTC counter. Clearing CNTEN has the effect of asynchronously resetting (synchronous reset negation) all the RTC and API logic as well as resetting the 32-bit counter. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues. 0 Counter disabled and reset 1 Counter enabled
RTCIE	RTC Interrupt Enable The RTCIE bit enables interrupts requests to the system if RTCF is asserted. 0 RTC interrupts disabled 1 RTC interrupts enabled
FRZEN	Freeze Enable Bit If RTCC[FRZEN] is set, the counter will stop on the last valid count when the device enters debug mode. On debug mode exit, the counter will resume counting from the frozen value. 0 Counter running in debug mode. 1 Counter stops (freezes) in debug mode.

Table 127. RTCC register 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. 0 RTC rollover wakeup/interrupt disabled 1 RTC rollover wakeup/interrupt enabled
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.
APIEN	Autonomous Periodic Interrupt Enable The APIEN bit enables the autonomous periodic interrupt function. 0 API disabled 1 API enabled
APIIE	API Interrupt Enable The APIIE bit enables interrupts requests to the system if APIF is asserted. 0 API interrupts disabled 1 API interrupts enabled
CLKSEL	Clock Select The CLKSEL[0:1] bits select 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 FXOSC
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 bits are added as an offset to the current RTC counter value which is then compared against the RTC counter. When there is a match, an API wakeup event occurs and if APIIE is set, an interrupt is also raised to the core. APIVAL can only be updated when APIEN is 0. **Note:** API functionality start only when APIVAL is non zero. The first API interrupt takes two more cycles because of synchronization of APIVAL to rtc clock and API_VAL+1 cycles for the subsequent occurences. After that interrupts are periodic in nature. Minimum value of APIVAL supported is 4. This is due to synchronization issues

13.4.3 RTC Status Register (RTCS)

The RTCS register contains:

- RTC interrupt flag
- API interrupt flag
- ROLLOVR Flag

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

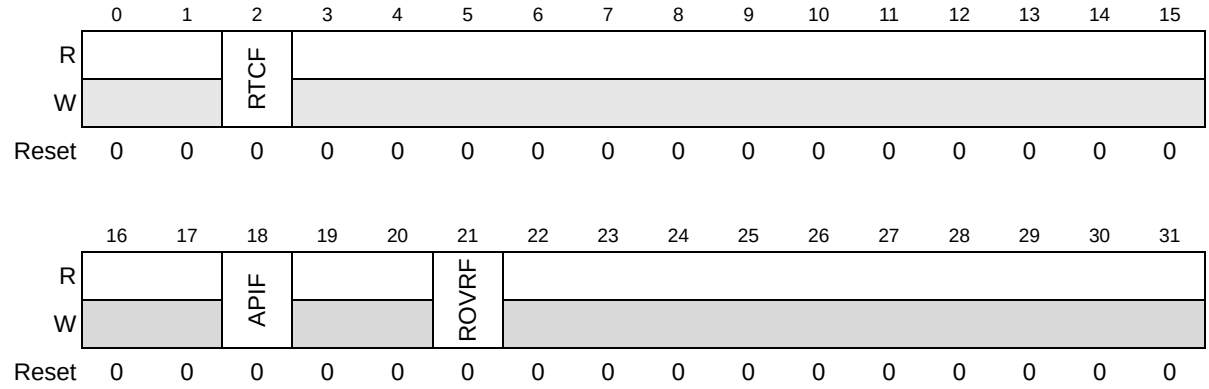


Figure 128. RTC Status Register (RTCS)

Table 128. RTCS register 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. 0 RTC counter is not equal to RTCVAL 1 RTC counter matches 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. 0 No API interrupt 1 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. 0 RTC has not rolled over 1 RTC has rolled over

13.4.4 RTC Counter Register (RTCCNT)

The RTCCNT register contains the current value of the RTC counter.

Address: RTC_BASE + 0x000C Access: User read

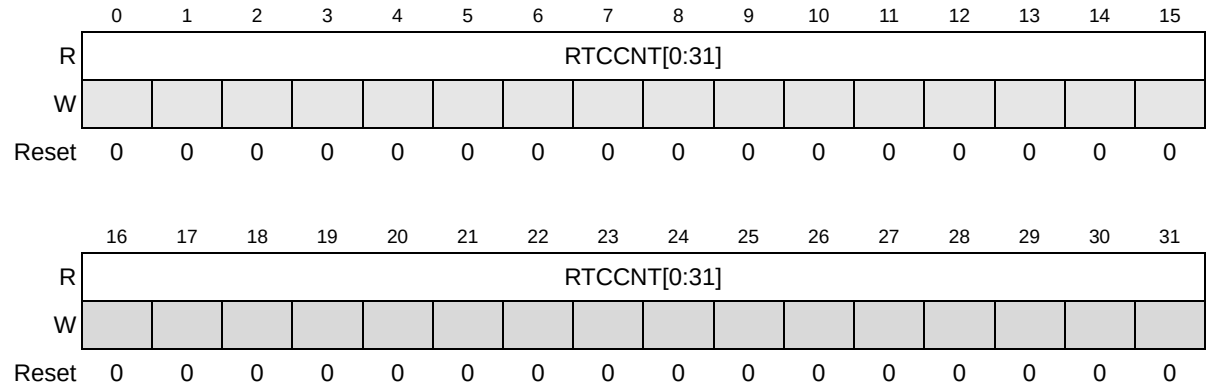


Figure 129. RTC Counter Register (RTCCNT)

Table 129. RTCCNT register field descriptions

Field	Description
RTCCNT	RTC Counter Value Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value.

13.5 RTC functional description

The RTC consists of a 32-bit free running counter enabled with the RTCC[CNTEN] bit. (When CNTEN is cleared, the counter is asynchronously reset.) 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 ipg_clk. 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 four options for clocking the RTC/API. The four clock sources are:

- The 16Mhz FIRC
- 4–40 Mhz fast external oscillator
- 128 Khz FIRC
- 32 Khz slow external oscillator

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. For example, the RTC supports interrupt requests in the range of 1 s to 4096 s (> 1 hr) with a 1 s

resolution, assuming a 32 kHz clock source with the divide by 32 enabled. 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. Rollover events are enabled by setting the RTCC[ROVEN] bit. If this bit is set, an RTC rollover will cause a wakeup request. To enable interrupts on an rollover request, the RTCC[RTCIE] bit also needs to be set.

All the flags and counter values are synchronized with ipg_clk. It is assumed that ipg_clk frequency is always more than or equal to the rtc_clk used to run the counter.

13.6 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 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 STANBY) or in RUN mode with crystal switched off.

Depending on both CAN baudrate 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 6 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 divided 16 MHz fast internal RC oscillator and the 5-bit clock prescaler.

After completion, software must process the sampled data in order to rebuild the 48 minimal bits.

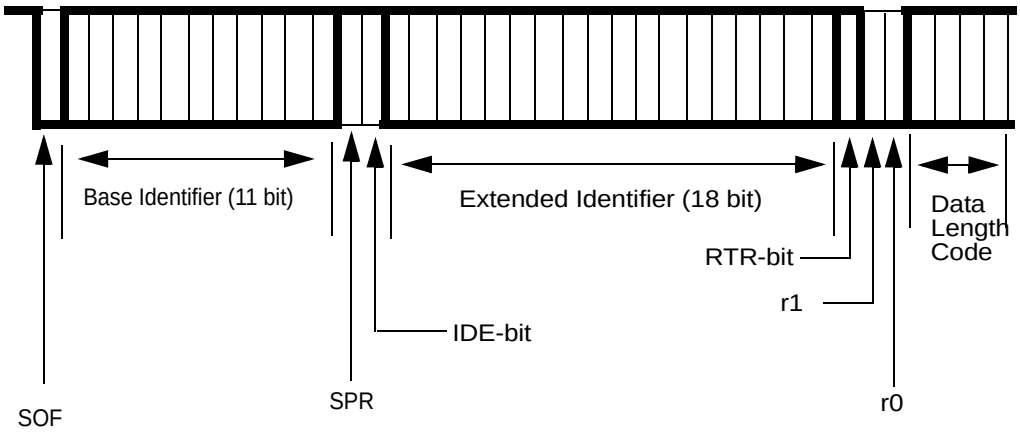


Figure 130. Extended CAN data frame

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 baudrates of 62.5 Kbit/s to 2 Mbit/s
- User selectable CAN Rx sample port [CAN0RX-CAN5RX]
- Divided 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 Memory map and register description

The CAN Sampler registers are listed in *Table 130*.

Table 130. CAN Sampler memory map

Base address: 0xFFE7_0000		
Address offset	**Register**	**Location**
0x00	Control Register (CR)	*on page 342*
0x04–0x30	Sample registers 0–11	*on page 343*

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 131. Control Register (CR)

Table 131. CR field descriptions

Field	Description
RX_COMPLETE	0: CAN frame has not been stored in the sample registers 1: CAN frame is stored in the sample registers
BUSY	This bit indicates the status of sampling 0: Sampling is complete or has not started 1: Sampling is ongoing
ACTIVE_CK	This bit indicates which is current clock for sample registers i.e xmem_ck. 0: ipg_clk_s is currently xmem_ck 1: RC_CLK is currently xmem_ck

Table 131. CR field descriptions (continued)

Field	Description
MODE	0:Skip the first frame and sample and store the second frame (SF_MODE) 1:Sample and store the first frame (FF_MODE)
CAN_RX_SEL	These bits determine which RX bit is sampled. 000: Rx0 is selected 001: Rx1 is selected 010: Rx2 is selected 011: Rx3 is selected 100: Rx4 is selected 101: Rx5 is selected
BRP	Baudrate Prescaler These bits are used to set the baudrate before going into standby mode 00000: Prescaler has 1 11111: Prescaler has 32
CAN_SMPLR_EN (1)	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

1. When CAN Sampler is enabled (i.e CR[CAN_SAMPLER_EN] = '1') and the peripheral is stopped by a mode transition through Mode Entry (MC_ME) block, it remains in the stopped state even after exiting the mode. The peripheral is unavailable for further sampling until and unless, either a device reset occurs or the CAN Sampler is disabled and enabled again. If the CAN Sampler has to be used again for sampling, it needs to be disabled and enabled again by programming CR[CAN_SAMPLER_EN] = '0' followed by CR[CAN_SAMPLER_EN] = '1'."

14.3.2 CAN Sampler Sample Registers 0–11

Offsets: 0x04–0x30 (12 registers) Access: Read/write

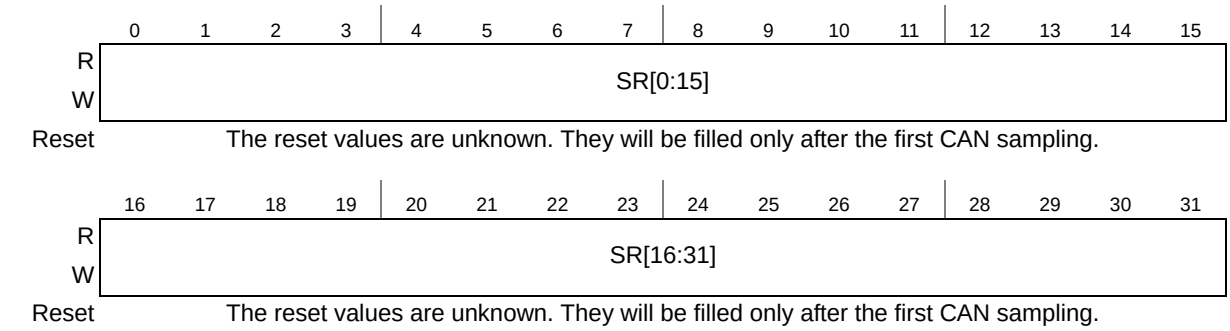


Figure 132. CAN Sampler Sample Registers 0–11

14.4 Functional description

As the CAN Sampler is driven by the divided 16 MHz fast internal RC oscillator to sample properly the CAN identifier, two modes are possible depending on both CAN baudrate and Low Power mode used:

- Immediate sampling on falling edge detection (first CAN frame): this mode is used when the divided 16 MHz fast internal RC oscillator is available in LP mode, e.g. STOP or HALT.
- Sampling on next frame (second CAN frame): this mode is used when the divided 16 MHz fast internal RC oscillator is switched off in LP mode, e.g. STANDBY. Due to the start-up times of both the Voltage regulator and the divided 16 MHz fast internal RC oscillator (~10µs), the CAN sampler would miss the first bits of a CAN identifier sent at 500 kbit/s. Therefore the first identifier is ignored and the sampling is performed on the first falling edge of after interframe space.

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 baudrate 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 CAN_SMPLR_EN bit in 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 RC oscillator. 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 RC 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 RC 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. 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 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 CAN_SMPLR_EN bit in the Control Register (CR) by writing ‘1’ to this bit.

Any activity on selected Rx line, the sampler enables the divided 16 MHz fast internal RC oscillator. When CAN_SMPLR_EN is reset to 0, the sampler should at least receive three RC clock pulses to reset itself, after which the RC can be switched off.

When the software wishes 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 vice versa, the sample register signals must be static and inactive to ensure the data is not corrupt.

14.4.2 Selecting the Rx port

One Rx port can be selected per sampling routine, which port to be sampled is selected by CAN_RX_SEL.

Table 132. Internal multiplexer correspondence

CAN_RX_SEL	Rx selected
000	CAN0RX on PB[1]
001	CAN1RX on PC[11]
010	CAN2RX on PE[9]
011	CAN3RX on PE[9]
100	CAN4RX on PC[11]
101	CAN5RX on PE[0]

14.4.3 Baudrate generation

Sampling is performed at a baudrate 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). User must set the baudrate prescaler (BRP) such that 8 samples per bit are achieved.

Baudrate setting must be made by software before going into standby or stop mode. This is done by setting BRP bits 5:1 in 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.

- Max. bitrate supported for sampling is 2 Mbit/s using BRP as 1
- Min. bitrate supported for sampling is 62.5 kbit/s using BRP as 32

For example, suppose system is transmitting at 125 kbit/s. 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 =1µs

BRP = 1µs/62.5ns = 16. Thus in this case BRP = 01111

15 e200z0h Core

15.1 Overview

The e200z0h processor family is a set of CPU cores that implement cost-efficient versions of the Power Architecture® technology. These processors are designed for deeply embedded control applications which require low cost solutions rather than maximum performance.

The e200z0h processor integrates an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable 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 built on Power Architecture technology with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in GPRs.

Instead of the base Power Architecture Book E instruction set support, the e200z0h core only implements the VLE (variable-length encoding) APU, providing improved code density. The VLE APU is further documented in the *Power Architecture VLE APU Definition*, a separate document.

15.2 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).
- 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

15.2.1 Microarchitecture summary

The 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), a 8 x 32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Prefetched instructions are placed into an instruction buffer with 4 entries, 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 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. 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.

15.2.1.1 Block diagram

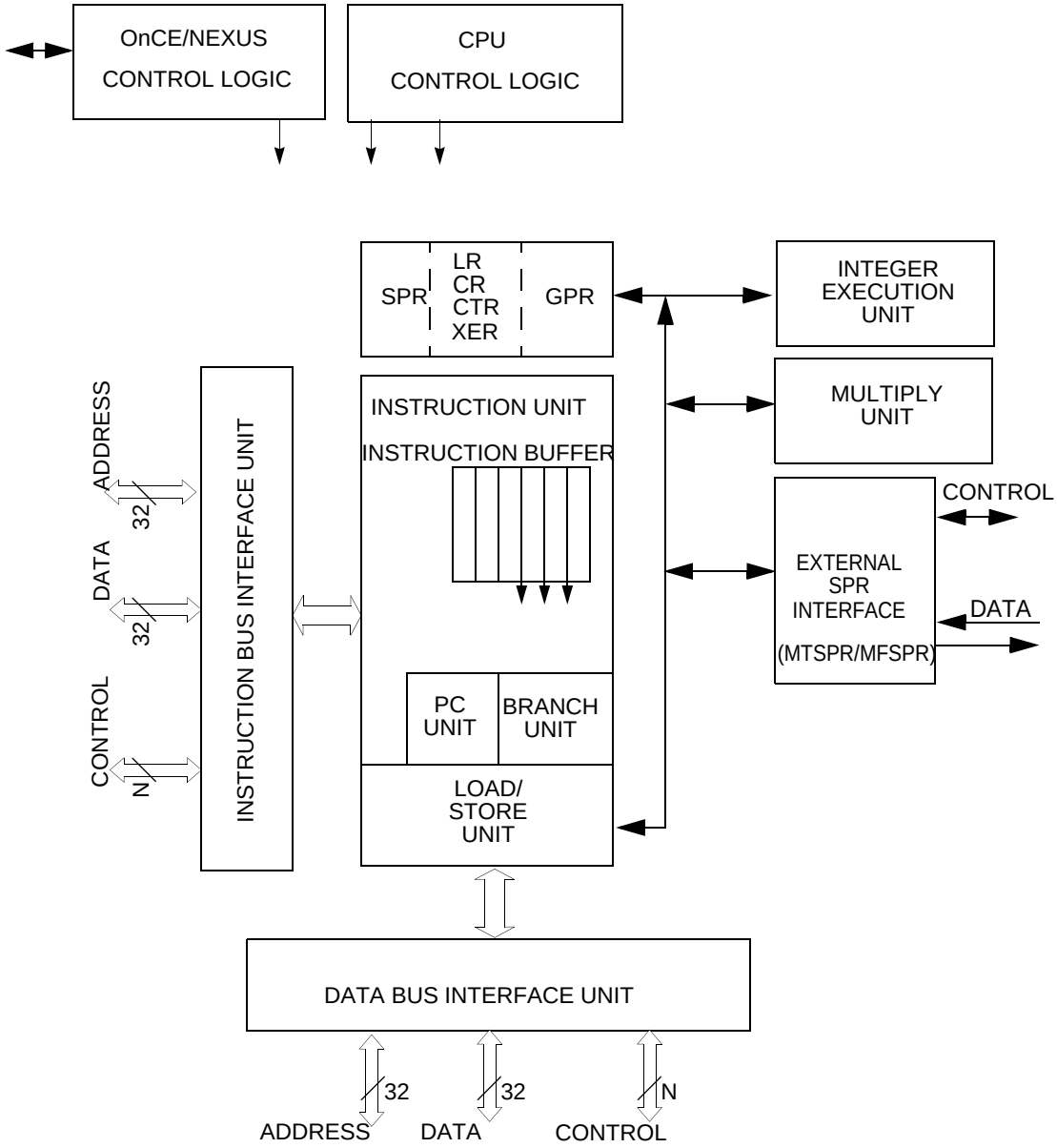


Figure 133. e200z0h block diagram

15.2.1.2 Instruction unit features

The features of the e200z0h Instruction unit are:

- 32-bit instruction fetch path supports fetching of 32-bit instruction per clock, or upto two16-bit VLE instructions per clock
- Instruction buffer with 4each 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

15.2.1.3 Integer unit features

The e200z0h 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)

15.2.1.4 Load/Store unit features

The e200z0h 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
- Dedicated 32-bit interface to memory

15.2.1.5 e200z0h System Bus features

The features of the e200z0h System Bus interface are as follows:

- Independent Instruction and Data Buses
- AMBA AHB2.v6 protocol
- 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

15.2.1.6 Nexus3+ features

The Nexus3+ module is compliant with Class 3 of the IEEE-ISTO 5001-2008 standard, with additional Class 4 features available. See *Section 42.2.1: NDI Features*.

15.3 Core registers and programmer's model

This section describes the registers implemented in the e200z0h core. It includes an overview of registers defined by the Power Architecture Book E architecture, highlighting differences in how these registers are implemented in the e200z0h core, and provides a detailed description of e200z0h-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Book E Specification.

The Power Architecture Book E 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 134 and *Figure 135* show the e200z0h 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 Book E 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.

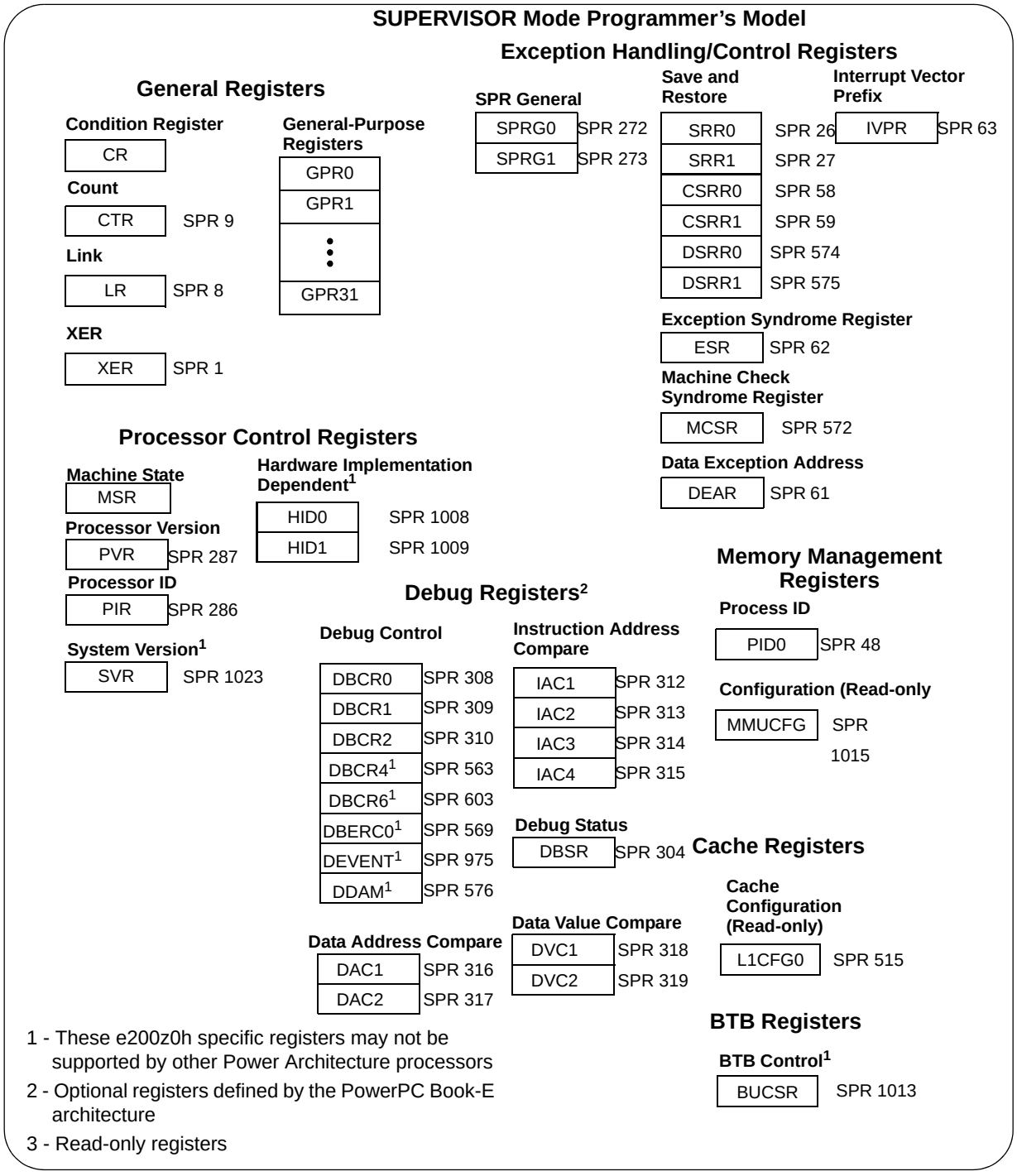


Figure 134. e200z0h Supervisor mode programmer's model

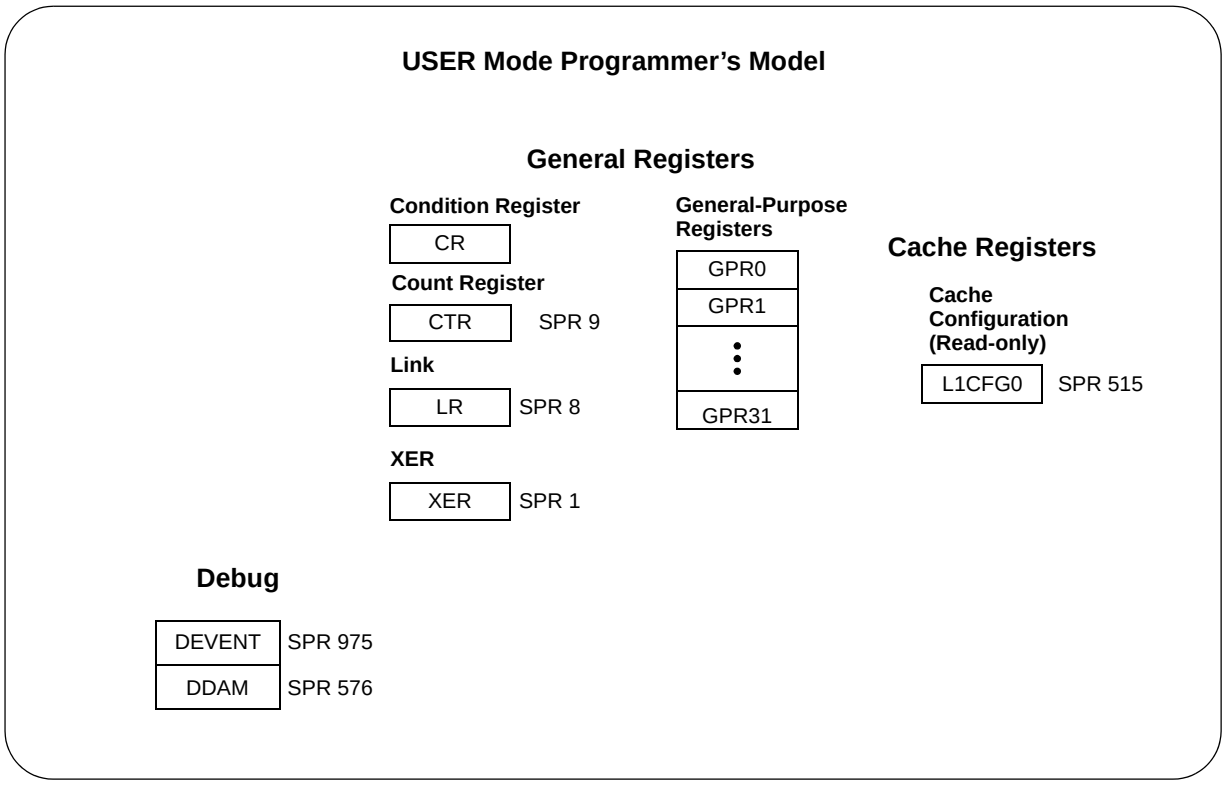


Figure 135. e200z0h User mode program model

General purpose registers (GPRs) are accessed through instruction operands. Access to other registers can be explicit (by using instructions for that purpose such as Move to Special Purpose Register (**mtspr**) and Move from Special Purpose Register (**mfspr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

15.3.1 Unimplemented SPRs and read-only SPRs

e200z0h fully decodes the SPR field of the **mfspr** and **mtspr**a instructions. **If the SPR specified is undefined and not privileged, an illegal instruction exception** is generated**.** If the SPR specified is undefined and privileged and the CPU is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is undefined and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

For the **mtspr** instruction, if the SPR specified is read-only **and not privileged, an illegal instruction exception** is generated**.** If the SPR specified is read-only and privileged and the core is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is read-only and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

For e200z0h, the following SPRs are not implemented and attempted access via a **mtspr** or **mfspr** instruction will result in an unimplemented instruction exception, unless the register is

privileged and the access attempt is made in user mode, in which case a privileged instruction exception will occur.

Table 133. List of Unimplemented SPRs

Type	Name
TImebase	**DEC, DECAR, TCR, TSR, TBU, TBL**
Software-Use Special Purpose Registers	USPRG0, SPRG2-7
Interrupt Vector Offset Registers	IVOR0-15(1)

1. These SPRs are hardwired to specific values, and are readable, but a mtspr will result in an unimplemented or privileged exception.

15.4 Instruction summary

e200z0h supports all VLE instructions described in the *Power Architecture VLE APU Definition version 1.2 together with the additional instructions for context save/restore.*

16 e200z4d Core

This chapter provides an overview of the e200z4d microprocessor core present in this device. It includes the following:

- An overview of the core, including the block diagram (*Figure 136*)
- A summary of the feature set for this core (see *Section 16.1: Features*)
  - A description of the execution units (see *Section 16.1.1: Execution Unit Features*)
  - A description of the memory management architecture (see *Section 16.1.3: Memory management unit features*)
  - High-level details of the core memory and coherency model (see *Section 16.1.4: External core complex interface features*)
  - High-level details of the Nexus 3+ features (see *Section 16.1.5: Nexus 3+ features*)
- A summary of the programming model for this core (see *Section 16.2: Programming model*)
  - An overview of the register set (see *Section 16.2.1: Register set*)
  - An overview of the instruction set (see *Section 16.2.2: Instruction set*)
  - An overview of interrupts and exception handling (see *Section 16.2.3: Interrupts and Exception Handling*)
- A summary of instruction pipeline and flow (see *Section 16.3: Microarchitecture summary*)

a) Overview

The e200z4d processor family is a set of CPU cores that implement low-cost versions of the Power Architecture technology.

The e200z4d is a dual-issue design based on the Power Architecture with 64-bit general purpose registers (GPRs). Power Architecture Book E floating-point instructions are not supported in hardware, but are trapped and may be emulated by software.

An Embedded Floating-point (EFPU) APU is provided to support real-time single-precision embedded numerics operations using the general-purpose registers.

A Signal Processing Extension (SPE) APU is provided to support real-time SIMD fixed point and single-precision, embedded numerics operations using the general-purpose registers. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). The GPRs have been extended to 64-bits in order to support vector instructions defined by the SPE APU. These instructions operate on a vector pair of 16-bit or 32-bit data types, and deliver vector and scalar results.

In addition to the base Power Architecture Book E instruction set support, the e200z4d core also implements the VLE (variable-length encoding) technology, providing improved code density. The VLE technology is further documented in “Power Architecture VLE Definition, Version 1.03", a separate document.

The e200z4d processor integrates a pair of integer execution units, a branch control unit, instruction fetch unit and load/store unit, and a multi-ported register file capable of sustaining six read and three 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 many cases.

The e200z4d contains a 4 Kb Instruction Cache as well as a Memory Management Unit. A Nexus Class 3+ module is also integrated.

Figure 136 shows the block diagram for the device.

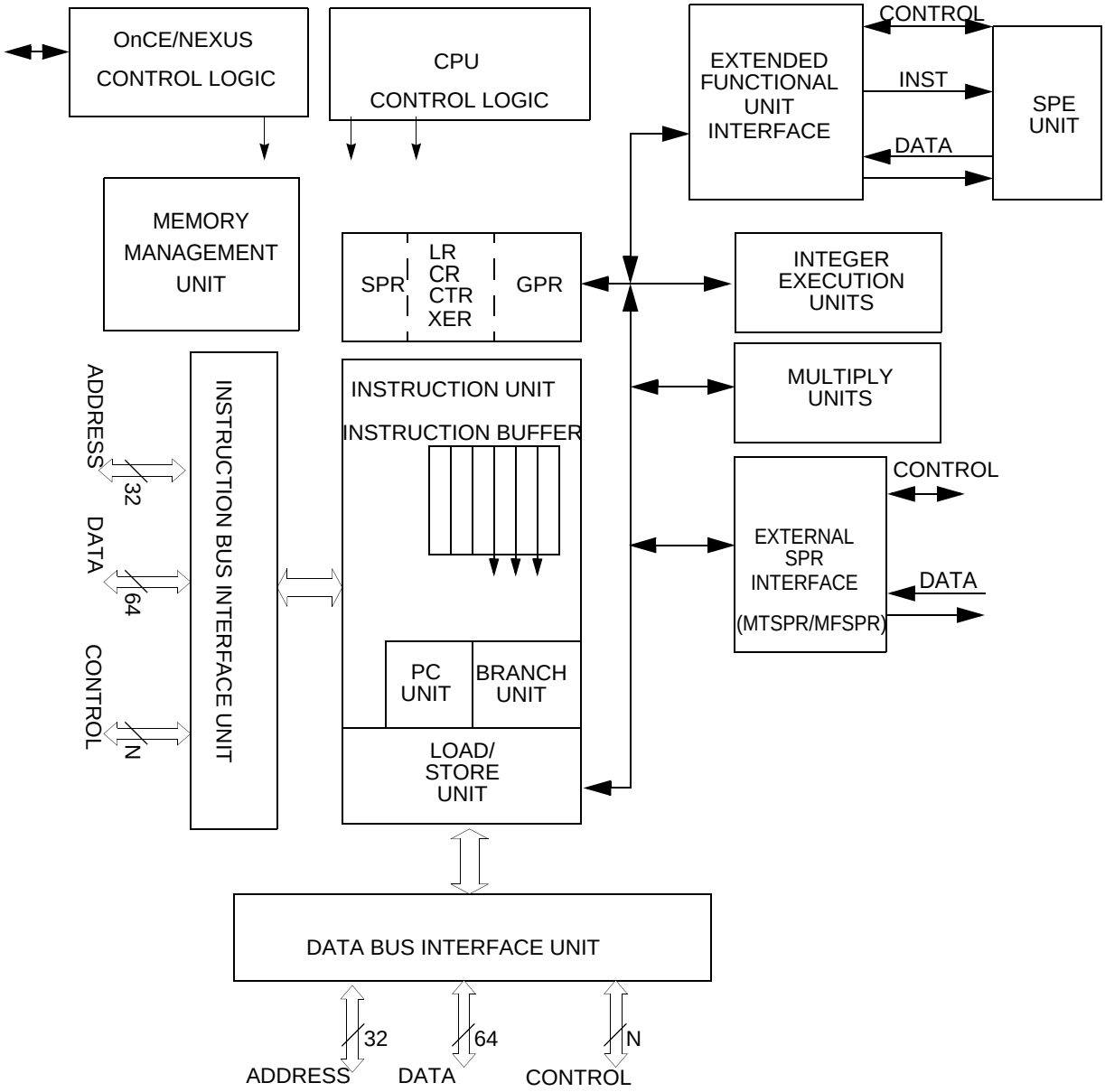


Figure 136. e200z4d Block Diagram

16.1 Features

Key features of the e200z4d are summarized as follows:

- Dual-issue, 32-bit Power ISA-compliant core
- Implementation of the VLE category for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit (BPU)
  - Dedicated branch address calculation adder
  - Branch target prefetching using an 8-entry branch target buffer (BTB)
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and flash memory by means of independent instruction and data bus interface units.
- Load/store unit
- 64-bit general-purpose register file
- Dual advanced high-performance (AHB) 2.v6 64-bit system buses
- Memory management unit (MMU) with 16-entry fully associative TLB and multiple page-size support
- 4 KB, 2/4-way set-associative instruction cache
- Signal Processing Extension (SPE1.1) APU supporting SIMD fixed-point operations using the 64-bit General Purpose Register file.
- Embedded Floating-Point (EFP2) APU supporting scalar and vector SIMD single-precision floating-point operations, using the 64-bit General Purpose Register file.
- Nexus Class 3+ real-time development unit
- Power management
  - Low power design—extensive clock gating
  - Power saving modes: nap, sleep, wait
  - Dynamic power management of execution units, cache, and MMU

See the following sections for more details about specific units.

16.1.1 Execution Unit Features

The following subsections describes the execution units main features.

16.1.1.1 Instruction Unit Features

The instruction unit features the following:

- 64-bit path to cache supports fetching of two 32-bit Power ISA instructions or four 16-bit VLE instructions per clock cycle.
- Instruction buffer holds up to eight 32-bit Power ISA instructions or sixteen 16-bit VLE instructions.
- Dedicated program counter (PC) incrementer supports instruction prefetches.
- Branch unit with dedicated branch address adder and branch target buffer supports single-cycle execution of successfully predicted branches.

16.1.1.2 Integer unit features

The integer units feature support for single-cycle execution of most integer instructions, as follows:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count-leading-zeros function
- 32-bit single-cycle barrel shifter for static shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in $\leq$ 14 clock cycles with minimized execution timing (Divide instruction will not be issued twice.)
- Pipelined 32 $\times$ 32 hardware multiplier array supports 32 $\times$ 32$\rightarrow$32 multiply with 2 clock latency, 1 clock throughput

16.1.1.3 Load/Store unit features

The load/store unit supports load, store, and load multiple/store multiple instructions by means of the following:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- Dedicated 64-bit interface to memory supports saving and restoring of up to two registers per cycle for load multiple and store multiple word instructions

16.1.2 L1 Cache features

The L1 cache features the following:

- 4 KB, 2- or 4-way configurable set-associative instruction cache
- 32-bit address bus plus attributes and control
- Supports Cache line locking
- Supports Way allocation
- Supports Tag and data parity or multi-bit EDC protection with correction/auto-invalidation capability
- Supports Tag and Data Double Error Detection
- Correction/Auto-invalidation capability

16.1.3 Memory management unit features

The memory management unit features the following:

- Virtual memory support
- 32-bit virtual and physical addresses
- 8-bit process identifier
- 16-entry fully associative TLB
- Hardware assist for TLB miss exceptions
- Per-entry multiple page size support from 1 Kbyte to 4 Gbyte
- Entry flush protection

16.1.4 External core complex interface features

The core complex interface features the following:

- Independent instruction and data buses
- Advanced microcontroller bus architecture (AMBA) AHB 2.v6 protocol
- 32-bit address bus, 64-bit data bus, plus attributes and control
- Data interface provides separate uni-directional 64-bit read and write data buses
- Support for HCLK running at a slower rate than CPU clock

16.1.5 Nexus 3+ features

The Nexus 3+ module provides real-time development capabilities for e200z4d processors in compliance with the IEEE-ISTO 5001-2010 standard. The '3+' suffix indicates that some Nexus Class 4 features are available. A portion of the pin interface (the JTAG port) is also shared with the OnCE/Nexus 1 unit.

The following features are implemented:

- Program trace by means of branch trace messaging.
  - 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.
- Data trace by means of data write messaging and data read messaging.
  - Provides the capability for the development tool to trace reads and/or writes to selected internal memory resources.
- Ownership trace by means of 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.
  - Allows enhanced download/upload capabilities.
- Data acquisition messaging
  - Allows code to be instrumented to export customized information to the Nexus auxiliary output port.
- Watchpoint messaging by means of the auxiliary interface
- Watchpoint trigger enable of program and/or data trace messaging
- Run-time access to the processor memory map by means of the JTAG port

All features are controllable and configurable by means of the JTAG port.

16.2 Programming model

This section describes the register model, instruction model, and the interrupt model as they are defined by the Power ISA, Freescale EIS, and the e200z4d implementation.

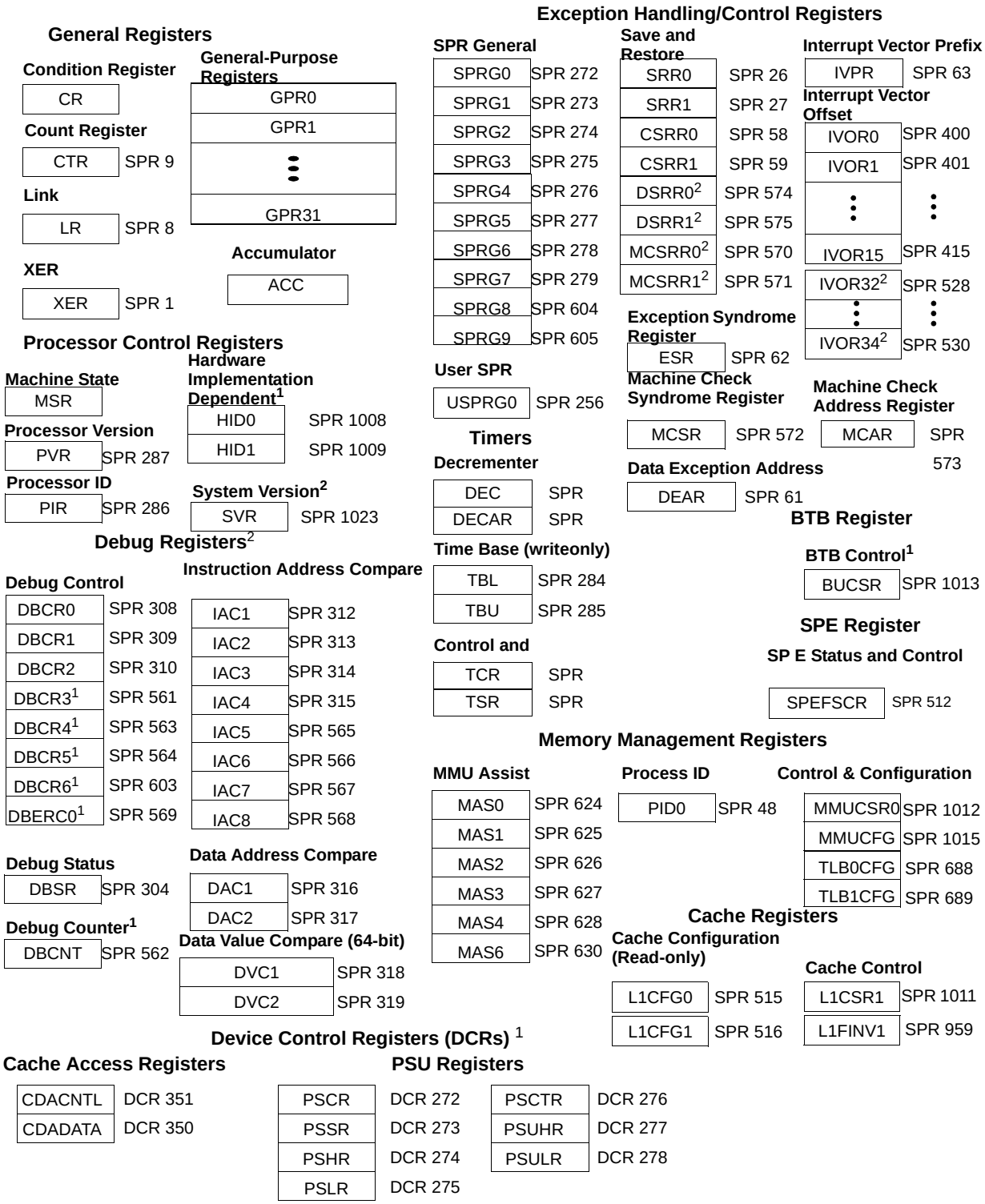
16.2.1 Register set

Figure 137 and *Figure 138* show the complete e200z4d register set, including the sets of the registers that are accessible in supervisor mode and the set of registers that are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal

number used in the instruction syntax to access the register. For example, the integer exception register (XER) is SPR 1.

Figure 137 shows the registers that can be accessed by supervisor-level software. User-level software can access only those registers listed in *Figure 138*.

Figure 137. e200z4d Supervisor Mode Programmer's Model



1 - These e200-specific registers may not be supported by other processors built on Power Architecture technology

2 - Optional registers defined by the Power ISA embedded architecture

3 - Read-only registers

Figure 138 shows the user-mode special-purpose registers.

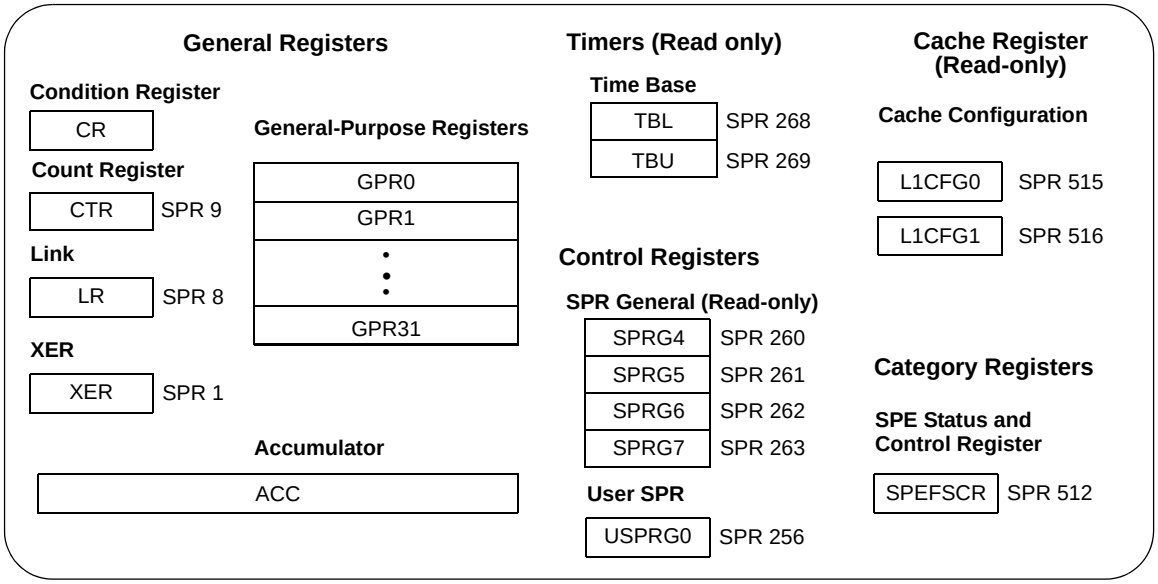


Figure 138. e200z4d User mode programmer's model SPRs

The GPRs are accessed through instruction operands. Access to other registers can be explicit, by using instructions for that purpose such as the Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions. Access to other registers can also be implicit, as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

16.2.2 Instruction set

The e200z4d supports the Power ISA instruction set for 32-bit embedded implementations. This is composed primarily of the user-level instructions defined by the user instruction set architecture (UISA). The e200z4d does not include the Power ISA floating-point, load string, or store string instructions.

The e200z4d core implements the following architectural extensions:

- The VLE category
- The integer select category (ISEL)
- Enhanced debug and the debug notify halt instruction categories
- The machine check category
- The WAIT category
- The volatile context save/restore category
- The embedded floating-point unit, version 2
- The signal processing extension unit, version 1.1
- The cache line locking category
- The enhanced reservations category

16.2.3 Interrupts and Exception Handling

The e200z4d core supports an extended exception handling model with nested interrupt capability and extensive interrupt vector programmability. In general, interrupt processing begins with an exception that occurs due to external conditions, errors, or program execution problems. When an exception occurs, the processor checks whether interrupt processing is enabled for that particular exception. If enabled, the interrupt causes the state of the processor to be saved in the appropriate registers and begins execution of the handler located at the associated vector address for that particular exception.

Once the handler is executing, the implementation may need to check bits in the exception syndrome register (ESR), the machine check syndrome register (MCSR), or the signal processing and embedded floating-point status and control register (SPEFSCR) to verify the specific cause of the exception and take appropriate action.

The core complex supports the interrupts described in *Table 134*.

Table 134. Interrupt Registers

Register	Description
Noncritical Interrupt Registers	
SRR0	Save/restore register 0—On noncritical interrupts, stores either the address of the instruction causing the exception or the address of the instruction that executes after the **rfi** instruction.
SRR1	Save/restore register 1—Saves machine state on noncritical interrupts and restores machine state after an **rfi** instruction is executed.
Critical Interrupt Registers	
CSRR0	Critical save/restore register 0—On critical interrupts, stores either the address of the instruction causing the exception or the address of the instruction that executes after the **rfci** instruction.
CSRR1	Critical save/restore register 1—Saves machine state on critical interrupts and restores machine state after an **rfci** instruction is executed.
Debug Interrupt Registers	
DSRR0	Debug save/restore register 0—On debug interrupts, stores either the address of the instruction causing the exception or the address of the instruction that executes after the **rfdi** instruction.
DSRR1	Debug save/restore register 1—Saves machine state on debug interrupts and restores machine state after an **rfdi** instruction is executed.
Machine Check Interrupts	
MCSRR0	Machine check save/restore register 0—On machine check interrupts, stores either the address of the instruction causing the exception or the address of the instruction that executes after the **rfmci** instruction.
MCSRR1	Machine check save/restore register 1—Saves machine state on machine check interrupts and restores those values when an **rfmci** instruction is executed
Syndrome Registers	
MCSR	Machine check syndrome register—Saves machine check syndrome information on machine check interrupts.
ESR	Exception syndrome register—Provides a syndrome to differentiate among the different kinds of exceptions that generate the same interrupt type. Upon generation of a specific exception type, the associated bits are set and all other bits are cleared.

Table 134. Interrupt Registers (continued)

Register	Description
SPE Interrupt Registers	
SPEFSCR	Signal processing and embedded floating-point status and control register—Provides interrupt control and status as well as various condition bits associated with the operations performed by the SPE. See *Table 135* for a list of the associated IVORs.
Other Interrupt Registers	
DEAR	Data exception address register—Contains the address that was referenced by a load, store, or cache management instruction that caused an alignment, data TLB miss, or data storage interrupt.
IVPR IVORs	Together, IVPR[32–47] || IVOR*n* [48–59] || 4bfr0000 define the address of an interrupt-processing routine. See *Table 135* for more information.
MSR	Machine state register—Defines the state of the processor. When an interrupt occurs, it is updated to preclude unrecoverable interrupts from occurring during the initial portion of the interrupt handler

Each interrupt has an associated interrupt vector address, obtained by concatenating IVPR[32–47] with the address index in the associated IVOR (that is, IVPR[32–47] || IVOR*n*[48–59] || 4b0000). The resulting address is that of the instruction to be executed when that interrupt occurs. IVPR and IVOR values are indeterminate on reset and must be initialized by the system software using **mtspr**.

Table 135 lists IVOR registers implemented on the e200z4d and the associated interrupts.

Table 135. Exceptions and conditions

IVOR*n*	Interrupt Type		IVOR*n*	Interrupt Type
None(1)	System reset (not an interrupt)		9	AP unavailable (not used by this core)
0(2)	Critical input		10	Decrementer
1	Machine check		11	Fixed-interval timer
	Machine check (non-maskable interrupt)		12	Watchdog timer
2	Data storage		13	Data TLB error
3	Instruction storage		14	Instruction TLB error
4(2)	External input		15	Debug
5	Alignment		16–31	Reserved
6	Program		32	SPE unavailable
7	Floating-point unavailable		33	SPE data exception
8	System call		34	SPE round exception

1. Vector to [*p_rstbase[0:29]*] || 2b00.
2. Autovectored external and critical input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

16.3 Microarchitecture summary

The e200z4d processor utilizes a five-stage pipeline for instruction execution. These stages operate in an overlapped fashion, allowing single clock-cycle instruction execution for most instructions. The stages are as follows:

1. Instruction fetch
2. Instruction decode/register file read/effective address calculation
3. Execute 0/memory access 0
4. Execute 1/memory access 1
5. Register write-back

The integer execution units consist of a 32-bit arithmetic unit, a logic unit, a 32-bit barrel shifter, a mask-insertion unit, a condition register manipulation unit, a count-leading-zeros unit, a 32×32 hardware multiplier array, and result feed-forward hardware. Integer unit 1 also supports hardware division.

Most arithmetic and logical operations are executed in a single cycle with the exception of multiply, which is implemented with a 2-cycle pipelined hardware array, and the divide instructions. A count-leading-zeros unit operates in a single clock cycle.

The instruction unit contains a program counter incrementer and 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 using the BTB is performed to accelerate taken branches. Prefetched instructions are placed into an 8-entry instruction buffer, with each entry capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Branch target addresses are calculated in parallel with branch instruction decode. Conditional branches that are not taken execute in a single clock cycle. Branches with successful BTB target prefetching have an effective execution time of one clock cycle if correctly predicted. All other taken branches have an execution time of two clock cycles.

Memory load and store operations are provided for byte, half-word, word (32-bit), and double-word data with automatic zero or sign extension of byte and half-word 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. There is a single load-to-use bubble for load instructions.

The condition register unit supports the condition register (CR) and condition register operations defined by the architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions. It also provides 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.

The SPE category supports vector instructions operating on 8-, 16-, and 32-bit fixed-point data types, as well as 32-bit IEEE Std. 754™ single-precision floating-point formats. It supports single-precision floating-point operations in a pipelined fashion.

The 64-bit general-purpose register file is used for source and destination operands, and there is a unified storage model for single-precision floating-point data types of 32-bits and the normal integer type. Low latency fixed-point and floating-point add, subtract, multiply,

multiply-add, multiply-sub, divide, compare, and conversion operations are provided. Most operations can be pipelined.

16.4 Availability of detailed documentation

Detailed documentation of the e200z4d core will be provided in a separate core reference manual (CRM). This CRM is available online at http://www.st.com.

17 Enhanced Direct Memory Access (eDMA)

17.1 Introduction

The enhanced direct memory access controller (eDMA) is a second-generation platform block capable of performing complex data movements through 32 programmable channels, with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation minimizes the overall block size.

Figure 139 is a block diagram of the eDMA module.

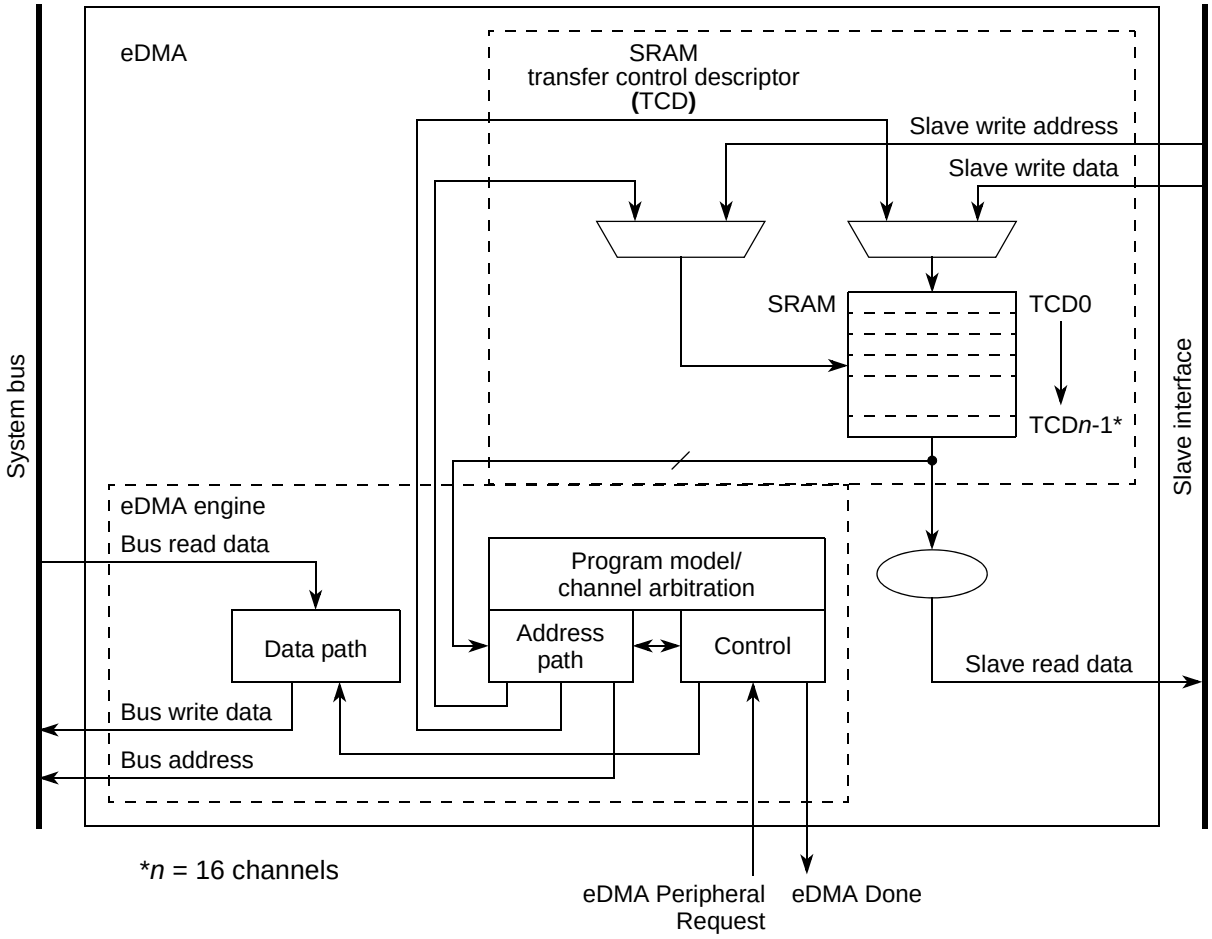


Figure 139. DMA block diagram

17.2 General features

The eDMA module supports the following features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An *inner* data transfer loop defined by a “minor” byte transfer count
  - An *outer* data transfer loop defined by a “major” iteration count
- Channel service request via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers – Independent channel linking at end of minor loop and/or major loop
  - Peripheral-paced hardware requests (one per channel)
  - For all three methods, *one service request per execution of the minor loop is required*
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Support for scatter/gather eDMA processing
- Support for complex data structures
- Support to cancel transfers via software or hardware

17.3 Device-specific features

- 32 programmable channels to support independent 8-, 16- or 32-bit single value or block transfers
- Support of variable sized queues and circular queues
- Source and destination address registers independently configured to post-incrementor remain constant
- Each transfer initiated by peripheral, CPU, periodic timer interrupt or eDMA channel request
- Peripheral eDMA request sources possible from DSPI, I^2C, 10-bit ADC, 12-bit ADC, LINFlexD, and eMIOS
- Each eDMA channel able to optionally send interrupt request to CPU on completion of single value or block transfer
- DMA transfers possible between system memories and all accessible memory mapped locations including peripheral and registers
- Programmable eDMA Channel Mux allows assignment of any eDMA source to any available eDMA channel with total of up to 64 request sources
- DMA supports the following functionality:
  - Scatter Gather
  - Channel Linking
  - Inner Loop Offset
  - Arbitration

    Fixed Group, fixed channel

    Round Robin Group, fixed channel

    Round Robin Group, Round Robin Channel

    Fixed Group, Round Robin Channel
  - Channel preemption
  - Cancel channel transfer
- Interrupts – The eDMA has a single interrupt request for each implemented channel and a combined eDMA Error interrupt to flag transfer errors to the system. Each channel eDMA interrupt can be enabled or disabled and provides notification of a completed transfer. Refer to the Interrupt Vector in the Interrupt Controller chapter of the reference manual for the allocation of these interrupts.

17.4 Memory map/register definition

The eDMA memory map is shown in *Table 136*. The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The eDMA's programming model is partitioned into two regions—the first region defines a number of registers providing control functions; the second region corresponds to the local transfer control descriptor memory.

Table 137 is a 32-bit view of the eDMA's memory map.

Table 136. eDMA memory map

Base address: 0xFFF4_4000		
Address offset	**Register**	**Location**
0x0000	EDMA_CR — eDMA control register	*on page 371*
0x0004	EDMA_ESR — eDMA error status register	*on page 373*
0x0008	Reserved	
0x000C	EDMA_ERQRL — eDMA enable request low register (channels 15–00)	*on page 375*
0x0010	Reserved	
0x0014	EDMA_EEIRL — eDMA enable error interrupt low register (channels 31–00)	*on page 376*
0x0018	EDMA_SERQR — eDMA set enable request register	*on page 377*
0x0019	EDMA_CERQR — eDMA clear enable request register	*on page 378*
0x001A	EDMA_SEEIR — eDMA set enable error interrupt register	*on page 378*
0x001B	EDMA_CEEIR — eDMA clear enable error interrupt register	*on page 378*
0x001C	EDMA_CIRQR — eDMA clear interrupt request register	*on page 379*
0x001D	EDMA_CER — eDMA clear error register	*on page 379*
0x001E	EDMA_SSBR — eDMA set start bit register	*on page 380*
0x001F	EDMA_CDSBR — eDMA clear done status bit register	*on page 380*
0x0020	Reserved	
0x0024	EDMA_IRQRL — eDMA interrupt request low register	*on page 381*
0x0028	Reserved	
0x002C	EDMA_ERL — eDMA error low register	*on page 382*
0x0030	Reserved	
0x0034	EDMA_HRSL — eDMA hardware request status register	*on page 383*
0x0038 – 0x01FF	Reserved	
0x0100–0x0111F	eDMA channel n priority register (EDMA_CPRn)	*on page 383*
0x0110	Reserved	
0x1000	TCD00 — eDMA transfer control descriptor 00	*on page 384*
0x1020	TCD01 — eDMA transfer control descriptor 01	*on page 384*
0x1040	TCD02 — eDMA transfer control descriptor 02	*on page 384*
0x1060	TCD03 — eDMA transfer control descriptor 03	*on page 384*
0x1080	TCD04 — eDMA transfer control descriptor 04	*on page 384*
0x10A0	TCD05 — eDMA transfer control descriptor 05	*on page 384*
0x10C0	TCD06 — eDMA transfer control descriptor 06	*on page 384*
0x10E0	TCD07 — eDMA transfer control descriptor 07	*on page 384*
0x1100	TCD08 — eDMA transfer control descriptor 08	*on page 384*

Table 136. eDMA memory map (continued)

Base address: 0xFFF4_4000		
Address offset	**Register**	**Location**
0x1120	TCD09 — eDMA transfer control descriptor 09	*on page 384*
0x1140	TCD10 — eDMA transfer control descriptor 10	*on page 384*
0x1160	TCD11 — eDMA transfer control descriptor 11	*on page 384*
0x1180	TCD12 — eDMA transfer control descriptor 12	*on page 384*
0x11A0	TCD13 — eDMA transfer control descriptor 13	*on page 384*
0x11C0	TCD14 — eDMA transfer control descriptor 14	*on page 384*
0x11E0	TCD15 — eDMA transfer control descriptor 15	*on page 384*
0x1200	Reserved	

Table 137. eDMA 32-bit memory map

DMA Offset	Register			
0x0000	DMA Control Register (EDMA_CR)			
0x0004	DMA Error Status (EDMA_ESR)			
0x0008	Reserved			
0x000C	DMA Enable Request Low (EDMA_ERQRL, Channels 31-00)			
0x0010	Reserved			
0x0014	DMA Enable Error Interrupt Low (EDMA_EEIRL, Channels 31-00)			
0x0018	DMA Set Enable Request (EDMA_SERQR)	DMA Clear Enable Request (EDMA_CERQR)	DMA Set Enable Error Interrupt (EDMA_SEEIR)	DMA Clear Enable Error Interrupt (EDMA_CEEIR)
0x001C	DMA Clear Interrupt Request (EDMA_CIRQR)	DMA Clear Error (EDMA_CER)	DMA Set Start Bit (EDMA_SSBR)	DMA Clear Done Status Bit (EDMA_CDSBR)
0x0020	Reserved			
0x0024	DMA Interrupt Request Low (EDMA_IRQRL, Channels 31-00)			
0x0028	Reserved			
0x002C	DMA Error Low (EDMA_ERL, Channels 31-00)			
0x0030	Reserved			
0x0034	DMA Hardware Request Status Low (EDMA_HRSL, Channels 31-00)			
0x0038 – 0x00FC	Reserved			
0x0100	DMA Channel 0 Priority (EDMA_CPR0)	DMA Channel 1 Priority (EDMA_CPR1)	DMA Channel 2 Priority (EDMA_CPR2)	DMA Channel 3 Priority (EDMA_CPR3)
0x0104	DMA Channel 4 Priority (EDMA_CPR4)	DMA Channel 5 Priority (EDMA_CPR5)	DMA Channel 6 Priority (EDMA_CPR6)	DMA Channel 7 Priority (EDMA_CPR7)
0x0108	DMA Channel 8 Priority (EDMA_CPR8)	DMA Channel 9 Priority (EDMA_CPR9)	DMA Channel 10 Priority (EDMA_CPR10)	DMA Channel 11 Priority (EDMA_CPR11)

Table 137. eDMA 32-bit memory map (continued)

DMA Offset	Register			
0x010C	DMA Channel 12 Priority (EDMA_CPR12)	DMA Channel 13 Priority (EDMA_CPR13)	DMA Channel 14 Priority (EDMA_CPR14)	DMA Channel 15 Priority (EDMA_CPR15)
0x0110	DMA Channel 16 Priority (EDMA_CPR16)	DMA Channel 17 Priority (EDMA_CPR17)	DMA Channel 18 Priority (EDMA_CPR18)	DMA Channel 19 Priority (EDMA_CPR19)
0x0114	DMA Channel 20 Priority (EDMA_CPR20)	DMA Channel 21 Priority (EDMA_CPR21)	DMA Channel 22 Priority (EDMA_CPR22)	DMA Channel 23 Priority (EDMA_CPR23)
0x0118	DMA Channel 24 Priority (EDMA_CPR24)	DMA Channel 25 Priority (EDMA_CPR25)	DMA Channel 26 Priority (EDMA_CPR26)	DMA Channel 27 Priority (EDMA_CPR27)
0x011c	DMA Channel 28 Priority (EDMA_CPR28)	DMA Channel 29 Priority (EDMA_CPR29)	DMA Channel 30 Priority (EDMA_CPR30)	DMA Channel 31 Priority (EDMA_CPR31)
0x0120 – 0x013C	Reserved			
0x0140 – 0x0FFC	Reserved			
0x1000 – 0x11FC	TCD00-TCD15			
0x1200-0x13fc	TCD16-TCD31			

17.4.1 Register descriptions

17.4.1.1 DMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

Arbitration among the channels can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see *Section 17.4.1.16: DMA Channel n Priority (EDMA_CPRn)*). In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 31 down to channel 0, without regard to priority.

See *Figure 140* and *Table 138* for the EDMA_CR definition.

Register address: EDMA_Offset + 0x0000 (EDMA_CR)

	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	CX	ECX
W																
RESET:	0	0	0	0	0	0	0	0	0	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	GRP1PRI		GRP0PRI		EMLM	CLM	HALT	HOE	ERGA	ERCA	EDBG	EBW
W																
RESET:	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 140. DMA Control Register (EDMA_CR)

Table 138. DMA Control Register (EDMA_CR) field descriptions

Name	Description	Value
CX	Cancel Transfer	0 Normal operation. 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CXFR bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.
ECX	Error Cancel Transfer	0 Normal operation. 1 Cancel the remaining data transfer in the same fashion as the CX cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the EDMA_ESR register and generating an optional error interrupt (see *Section 17.4.1.2: DMA Error Status (EDMA_ESR)*).
GRP1PRI	Channel Group 1 Priority	Group 1 priority level when fixed priority group arbitration is enabled.
GRP0PRI	Channel Group 0 Priority	Group 0 priority level when fixed priority group arbitration is enabled.
EMLM	Enable Minor Loop Mapping	0 Minor loop mapping disabled. TCDn.word2 is defined as a 32-bit nbytes field. 1 Minor loop mapping enabled. When set, TCDn.word2 is redefined to include individual enable fields, an offset field and the nbytes field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The nbytes field is reduced when either offset is enabled.
CLM	Continuous Link Mode	0 A minor loop channel link made to itself will go through channel arbitration before being activated again. 1 A minor loop channel link made to itself will not go through channel arbitration before being activated again. Upon minor loop completion the channel will active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
HALT	Halt DMA Operations	0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution will resume when the HALT bit is cleared.
HOE	Halt On Error	0 Normal operation. 1 Any error will cause the HALT bit to be set. Subsequently, all service requests will be ignored until the HALT bit is cleared.

Table 138. DMA Control Register (EDMA_CR) field descriptions (continued)

Name	Description	Value
ERGA	Enable Round Robin Group Arbitration	0 Fixed priority arbitration is used for selection among the groups. 1 Round robin arbitration is used for selection among the groups.
ERCA	Enable Round Robin Channel Arbitration	0 Fixed priority arbitration is used for channel selection within each group. 1 Round robin arbitration is used for channel selection within each group.
EDBG	Enable Debug	0 The assertion of the device debug mode is ignored. 1 The assertion of the device debug mode causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution will resume when either the device comes out of debug mode or the EDBG bit is cleared.
EBW	Enable Buffered Writes	0 The bufferable write signal (hprot[2]) is not asserted during AMBA AHB writes. 1 The bufferable write signal (hprot[2]) is asserted on all AMBA AHB writes except for the last write sequence.

17.4.1.2 DMA Error Status (EDMA_ESR)

The EDMA_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit is not equal to the TCD.BITER.E_LINK bit. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write will execute using the data

captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence will execute before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Register address: EDMA_Offset + 0x0004 (EDMA_ESR)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CPE	ERRCHN[0:5]						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 141. DMA Error Status (EDMA_ESR) Register

Table 139. DMA Error Status (EDMA_ESR) field descriptions

Name	Description	Value
VLD	Logical OR of all EDMA_ERL status bits.	0 No EDMA_ERL bits are set. 1 At least one EDMA_ERL bit is set indicating a valid error exists that has not been cleared.
CPE	Channel Priority Error	0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group. All channel priorities within a group are not unique.
ERRCHN[0:5]	Error Channel Number or Cancelled Channel Number	The channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled.
SAE	Source Address Error	0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.saddr field. TCD.saddr is inconsistent with TCD.ssize.
SOE	Source Offset Error	0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.soff field. TCD.soff is inconsistent with TCD.ssize.
DAE	Destination Address Error	0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.daddr field. TCD.daddr is inconsistent with TCD.dsize.

Table 139. DMA Error Status (EDMA_ESR) field descriptions (continued)

Name	Description	Value
DOE	Destination Offset Error	0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.doff field. TCD.doff is inconsistent with TCD.dsize.
NCE	Nbytes/Citer Configuration Error	0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the TCD.nbytes or TCD.citer fields. TCD.nbytes is not a multiple of TCD.ssize and TCD.dsize, or TCD.citer is equal to zero, or TCD.citer.e_link is not equal to TCD.biter.e_link.
SGE	Scatter/Gather Configuration Error	0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.dlast_sga field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.e_sg is enabled. TCD.dlast_sga is not on a 32 byte boundary.
SBE	Source Bus Error	0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error	0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

17.4.1.3 DMA Enable Request (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 32channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 31–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR, and EDMA_CERQR registers. The EDMA_CERQR and EDMA_SERQR registers are provided so the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA_ERQRL register.

Both the eDMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made through software or a linked channel request.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ31	ERQ30	ERQ29	ERQ28	ERQ27	ERQ26	ERQ25	ERQ24	ERQ23	ERQ22	ERQ21	ERQ20	ERQ19	ERQ18	ERQ17	ERQ16
RESET:	0	0	0	0	0	0	0	0	0	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	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ09	ERQ08	ERQ07	ERQ06	ERQ05	ERQ04	ERQ03	ERQ02	ERQ01	ERQ00
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Register address: EDMA_Offset +0x000C (EDMA_ERQRL)

Figure 142. DMA Enable Request (EDMA_ERQRL) Registers

Table 140. DMA Enable Request (EDMA_ERQRL) field descriptions

Name	Description	Value
ERQn	Enable eDMA Request n	0 The eDMA request signal for channel n is disabled. 1 The eDMA request signal for channel n is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA_ERQRL bit for that channel. If the TCD.d_req bit is set, then the corresponding EDMA_ERQRL bit is cleared, disabling the eDMA request; else if the d_req bit is cleared, the state of the EDMA_ERQRL bit is unaffected.

17.4.1.4 DMA Enable Error Interrupt (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 32 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 31–0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR registers. The EDMA_SEEIR and EDMA_CEEIR registers are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA_EEIRL register.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See *Figure 143* and *Table 141* for the EDMA_EEIRL definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI31	EEI30	EEI29	EEI28	EEI27	EEI26	EEI25	EEI24	EEI23	EEI22	EEI21	EEI20	EEI19	EEI18	EEI17	EEI16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Register address: EDMA_Offset + 0x0014 (EDMA_EEIRL)

Figure 143. DMA Enable Error Interrupt (EDMA_EEIRL) Register

Table 141. DMA Enable Error Interrupt (EDMA_EEIRL) field descriptions

Name	Description	Value
EEIn	Enable Error Interrupt n	0 The error signal for channel n does not generate an error interrupt. 1 The assertion of the error signal for channel n generate an error interrupt request.

17.4.1.5 DMA Set Enable Request (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set function, forcing the entire contents of EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

Register address: EDMA_Offset + 0x0018 (EDMA_SERQR)

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SERQ[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 144. DMA Set Enable Request (EDMA_SERQR) Register

Table 142. DMA Set Enable Request (EDMA_SERQR) field descriptions

Name	Description	Value
SERQ[0:6]	Set Enable Request	0-31 Set the corresponding bit in EDMA_ERQRL 64-127 Set all bits in EDMA_ERQRL

17.4.1.6 DMA Clear Enable Request (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes. See *Figure 145* and *Table 143* for the EDMA_CERQR definition.

Register address: EDMA_Offset + 0x0019 (EDMA_CERQR)

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERQ[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 145. DMA Clear Enable Request (EDMA_CERQR) Register

Table 143. DMA Clear Enable Request (EDMA_CERQR) field descriptions

Name	Description	Value
CERQ[0:6]	Clear Enable Request	0-63 Clear corresponding bit in EDMA_ERQRL 64-127 Clear all bits in EDMA_ERQRL

17.4.1.7 DMA Set Enable Error Interrupt (EDMA_SEEIR)

The EDMA_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes. See *Figure 146* and *Table 144* for the EDMA_SEEIR definition.

Register address: EDMA_Offset + 0x001A (EDMA_SEEIR)

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SEEI[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 146. DMA Set Enable Error Interrupt (EDMA_SEEIR) Register

Table 144. DMA Set Enable Error Interrupt (EDMA_SEEIR) field descriptions

Name	Description	Value
SEEI[0:6]	Set Enable Error Interrupt	0-63 Set the corresponding bit in EDMA_EEIRL 64-127 Set all bits in EDMA_EEIRL

17.4.1.8 DMA Clear Enable Error Interrupt (EDMA_CEEIR)

The EDMA_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register

write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register returns all zeroes. See *Figure 147* and *Table 145* for the EDMA_CEEIR definition.

Register address: EDMA_Offset + 0x001B (EDMA_CEEIR)

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CEEI[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 147. DMA Clear Enable Error Interrupt (EDMA_CEEIR) Register

Table 145. DMA Clear Enable Error Interrupt (EDMA_CEEIR) field descriptions

Name	Description	Value
CEEI[0:6]	Clear Enable Error Interrupt	0-63 Clear corresponding bit in EDMA_EEIRL 64-127 Clear all bits in EDMA_EEIRL

17.4.1.9 DMA Clear Interrupt Request (EDMA_CIRQR)

The EDMA_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes. See *Figure 148* and *Table 146* for the EDMA_CIRQR definition.

Register address: EDMA_Offset + 0x001C

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CINT[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 148. DMA Clear Interrupt Request (EDMA_CIRQR) Fields

Table 146. DMA Clear Interrupt Request (EDMA_CIRQR) field descriptions

Name	Description	Value
CINT[0:6]	Clear Interrupt Request	0-63 Clear the corresponding bit in EDMA_IRQRL 64-127 Clear all bits in EDMA_IRQRL

17.4.1.10 DMA Clear Error (EDMA_CER)

The EDMA_CER provides a memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA_ERL to

be zeroed, clearing all channel error indicators. Reads of this register return all zeroes. See *Figure 149* and *Table 147* for the EDMA_CER definition.

Register address: EDMA_Offset + 0x001E (EDMA_CER)

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERR[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 149. DMA Clear Error (EDMA_CER) Register

Table 147. DMA Clear Error (EDMA_CER) field descriptions

Name	Description	Value
CERR[0:6]	Clear Error Indicator	0-63 Clear corresponding bit in EDMA_ERL 64-127 Clear all bits in EDMA_ERL

17.4.1.11 DMA Set START Bit (EDMA_SSBR)

The EDMA_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes. See *Table 155* for the TCD START bit definition.

Register address: EDMA_Offset + 0x001E

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SSRT[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 150. DMA Set START Bit (EDMA_SSBR) Register

Table 148. DMA Set START Bit (EDMA_SSBR) field descriptions

Name	Description	Value
SSRT[0:6]	Set START Bit (Channel Service Request)	0-63 Set the corresponding channel's TCD.start 64-127 Set all TCD.start bits

17.4.1.12 DMA Clear DONE Status (EDMA_CDSBR)

The EDMA_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared. See *Table 155* for the TCD DONE bit definition.

Register address: EDMA_Offset + 0x001F (EDMA_CDSBR)

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CDNE[0:6]						
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 151. DMA Clear DONE Status (EDMA_CDSBR) Register

Table 149. DMA Clear DONE Status (EDMA_CDSBR) field descriptions

Name	Description	Value
CDNE[0:6]	Clear DONE Status Bit	0-63 Clear the corresponding channel's DONE bit 64-127 Clear all TCD DONE bits

17.4.1.13 DMA Interrupt Request (EDMA_IRQRL)

The EDMA_IRQRL provides a bit map for the 32channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 31–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a single channel can be cleared without performing a read-modify-write sequence to the EDMA_IRQRL. See *Figure 152* and *Table 150* for the EDMA_IRQL definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INT15	INT14	INT13	INT12	INT11	INT10	INT09	INT08	INT07	INT06	INT05	INT04	INT03	INT02	INT01	INT00
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Register address: EDMA_Offset + 0x0024 (EDMA_IRQRL)

Figure 152. DMA Interrupt Request (EDMA_IRQRL) Registers

Table 150. DMA Interrupt Request (EDMA_IRQRL) field descriptions

Name	Description	Value
INTn	DMA Interrupt Request n	0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

17.4.1.14 DMA Error (EDMA_ERL)

The EDMA_ERL provides a bit map for the 32 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 31-0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIR, then logically summed across 32 channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIR. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CER. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no affect on the corresponding channel's current error status. The EDMA_CER is provided so the error indicator for a single channel can be cleared. See *Figure 153* and *Table 151* for the EDMA_ERL definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR31	ERR30	ERR29	ERR28	ERR27	ERR26	ERR25	ERR24	ERR23	ERR22	ERR21	ERR20	ERR19	ERR18	ERR17	ERR16
RESET:	0	0	0	0	0	0	0	0	0	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	ERR15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR09	ERR08	ERR07	ERR06	ERR05	ERR04	ERR03	ERR02	ERR01	ERR00
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Register address: EDMA_Offset + 0x002C (EDMA_ERL)

Figure 153. DMA Error (EDMA_ERL) Registers

Table 151. DMA Error (EDMA_ERL) field descriptions

Name	Description	Value
ERRn	DMA Error n	0 An error in channel n has not occurred. 1 An error in channel n has occurred.

17.4.1.15 DMA Hardware Request Status (EDMA_HRSL)

The EDMA_HRSL register provides a bit map for the implemented channels to show the current hardware request status for each channel. This view into the hardware request signals may be used for debug purposes.

See *Figure 154* and *Figure 152* for the EDMA_HRSL definition.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS31	HRS30	HRS29	HRS28	HRS27	HRS26	HRS25	HRS24	HRS23	HRS22	HRS21	HRS20	HRS19	HRS18	HRS17	HRS16
RESET:	0	0	0	0	0	0	0	0	0	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	HRS15	HRS14	HRS13	HRS12	HRS11	HRS10	HRS09	HRS08	HRS07	HRS06	HRS05	HRS04	HRS03	HRS02	HRS01	HRS00
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Register address: EDMA_Offset + 0x0034 (EDMA_HRSL)

Figure 154. DMA Hardware Request Status (EDMA_HRSL) Register

Table 152. DMA Hardware Request Status (EDMA_HRSL) field descriptions

Name	Description	Value
HRSn	DMA Hardware Request Status n	0 A hardware service request for channel n is not present. 1 A hardware service request for channel n is present. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[n] bit.

17.4.1.16 DMA Channel n Priority (EDMA_CPRn)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error will be reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPRn register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel requests service, the restored channel will be suspended and the higher priority channel will be serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a

preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected for channel arbitration mode

A channel's ability to preempt another channel can be disabled by setting the DPA bit in the EDMA_CPRn register. When a channel's preempt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel. See *Figure 155* and *Table 153* for the EDMA_CPRn definition.

Register address: EDMA_Offset + 0x100 + n (EDMA_CPRn)

	0	1	2	3	4	5	6	7
R	ECP	DPA	GRPPRI[0:1]		CHPRI[0:3]			
W								
RESET:	0	0	*	*	*	*	*	*

= Unimplemented,
* = defaults to channel number (n) after reset

Figure 155. DMA Channel n Priority (EDMA_CPRn) Register

Table 153. DMA Channel n Priority (EDMA_CPRn) field descriptions

Name	Description	Value
ECP	Enable Channel Preemption	0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
DPA	Disable Preempt Ability	0 Channel n can suspend a lower priority channel. 1 Channel n cannot suspend any channel, regardless of channel priority.
CHPRI[0:3]	Channel n Arbitration Priority	Channel priority when fixed-priority arbitration is enabled.

17.4.1.17 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as eight 32-bit values. *Table 154* is a field list of the basic TCD structure.

Table 154. TCDn 32-bit Memory Structure

eDMA Offset	TCDn Field	
0x1000+(32 x n)+0x0000	Source address (saddr)	
0x1000+(32 x n)+0x0004	Transfer attributes	Signed source address offset (soff)
0x1000+(32 x n)+0x0008	Inner minor byte count (nbytes)	
0x1000+(32 x n)+0x000C	Last source address adjustment (slast)	

Table 154. TCDn 32-bit Memory Structure (continued)

eDMA Offset	TCDn Field	
0x1000+(32 x n)+0x0010	Destination address (daddr)	
0x1000+(32 x n)+0x0014	Current major iteration count (citer)	Signed destination address offset (doff)
0x1000 (32 x n) 0x0018	Last destination address adjustment / scatter-gather address (dlast_sga)	
0x1000+(32 x n)+0x001c	Beginning major iteration count (biter)	Channel control/status

Figure 156 and *Table 155* define the fields of the TCD*n* structure.

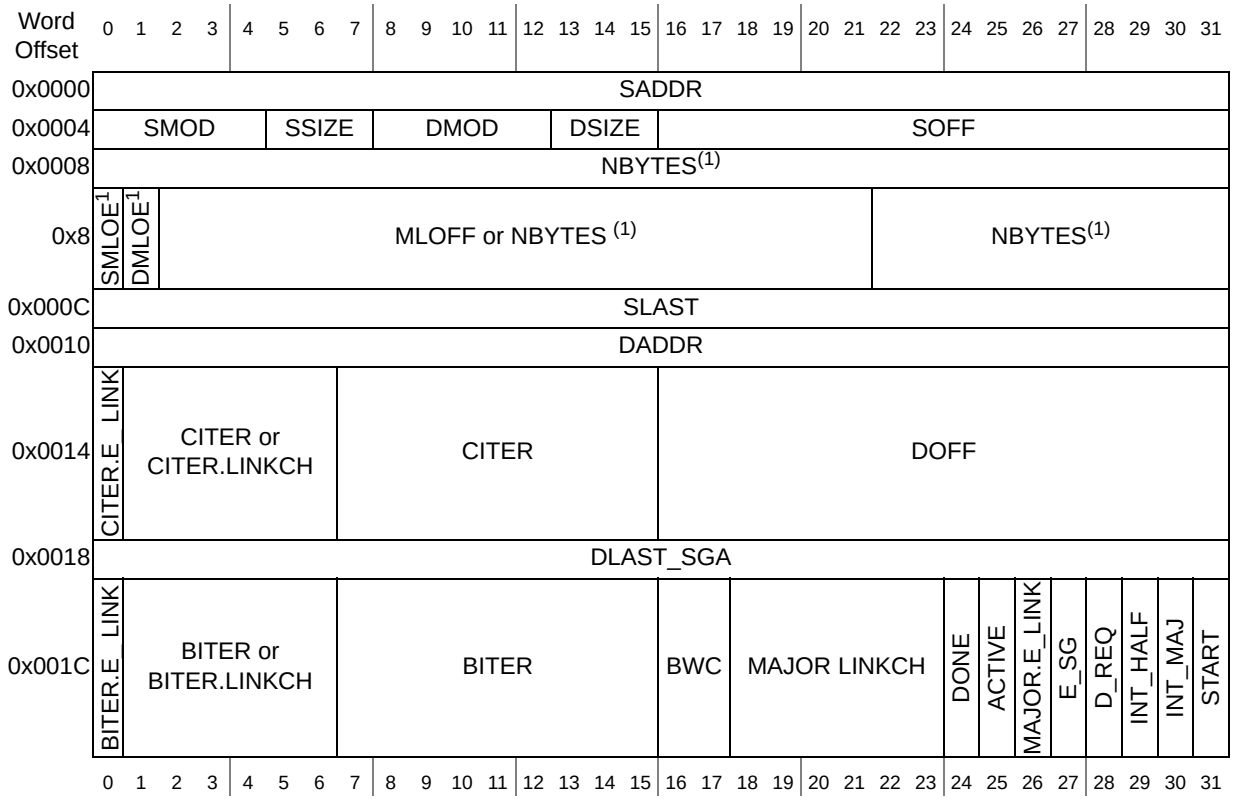


Figure 156. TCD Structure

1. The fields implemented in Word 2 depend on whether EDMA_CR(EMLM) is set to 0 or 1. Refer to *Table 138*.

Note: The TCD structures for the eDMA channels shown in Figure 156 are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 155. TCD*n* field descriptions

Bits / Word Offset [n:n]	Name	Description
0–31 / 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 / 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
37–39 / 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 Reserved 100 16-byte (32-bit, 4-beat, WRAP4 burst) 101 32-byte (32-bit, 8 beat, WRAP8 burst) 110 Reserved 111 Reserved The attempted specification of a reserved encoding causes a configuration error.
40–44 / 0x4 [8:12]	DMOD [0:4]	Destination address modulo. See the SMOD[0:5] definition.
45–47 / 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. See the SSIZE[0:2] definition.
48–63 / 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64–95 / 0x8 [0:31]	NBYTES(1) [0:31]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. **Note:** The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 Gbyte transfer.

Table 155. TCD*n* field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
64 0x8 [0]	SMLOE (1) 0	Source minor loop offset enable This flag selects whether the minor loop offset is applied to the source address upon minor loop completion. 0 The minor loop offset is not applied to the saddr. 1 The minor loop offset is applied to the saddr.
65 0x8 [1]	DMLOE (1) 1	Destination minor loop offset enable This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion. 0 The minor loop offset is not applied to the daddr. 1 The minor loop offset is applied to the daddr.
66–85 0x8 [2-21]	MLOFF or NBYTES (1) [0:19]	Inner “minor” byte transfer count or Minor loop offset If both SMLOE and DMLOE are cleared, this field is part of the byte transfer count. If either SMLOE or DMLOE are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.
86–95 / 0x8 [22:31]	NBYTES (1)	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. **Note:** The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GByte transfer.
96–127 / 0xC [0:31]	SLAST [0:31]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
128–159 / 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 / 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER.E_LINK bit otherwise a configuration error will be reported.

Table 155. TCD*n* field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
161–166 / 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field. Otherwise, – After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel's TCD.START bit.
167–175 / 0x14 [7:15]	CITER [6:14]	Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. **Note:** When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
176–191 / 0x14 [16:31]	DOFF [0:15]	Destination address signed Offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
192–223 / 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (TCD.E_SG = 0) then – Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise, – This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.

Table 155. TCD*n* field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
224 / 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
225–230 / 0x1C [1:6]	BITER [0:5] or BITER.LINKCH[0:5]	Starting major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field. Otherwise, – After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel's TCD.START bit. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 / 0x1C [7:15]	BITER [6:14]	Starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
240–241 / 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). 00 No DMA engine stalls 01 Reserved 10 DMA engine stalls for 4 cycles after each r/w 11 DMA engine stalls for 8 cycles after each r/w

Table 155. TCD*n* field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
242–247 / 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then, – No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise – After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 / 0x1C [24]	DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs). **Note:** This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 / 0x1C [25]	ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.
250 / 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. NOTE: To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 / 0x1C [27]	E_SG	Enable scatter-gather processing. As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. NOTE: To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.

Table 155. TCD*n* field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
252 / 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQRL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQRL bit is not affected. 1 The channel's EDMA_ERQRL bit is cleared when the outer major loop is complete.
253 / 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQRL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled. **Note:** If BITER = 1, do not use INT_HALF; use INT_MAJ instead.
254 / 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQRL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 / 0x1C [31]	START	Channel start. If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

1. The fields implemented at 0x8 depend on whether EDMA_CR(EMLM) is set to 0 or 1. Refer to *Table 138*.

17.5 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.

The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally

enabled by EDMA_CPR*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCD*n*.{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter-gather operation.

– Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.

  The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).

– Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).

– Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved.

  A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as:

  if (SSIZE < DSIZE)

  transfer size = destination transfer size (# of bytes)

  else

  transfer size = source transfer size (# of bytes)

  Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

  For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  – Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
  – Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

17.5.1 eDMA Basic data flow

The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 157*, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.

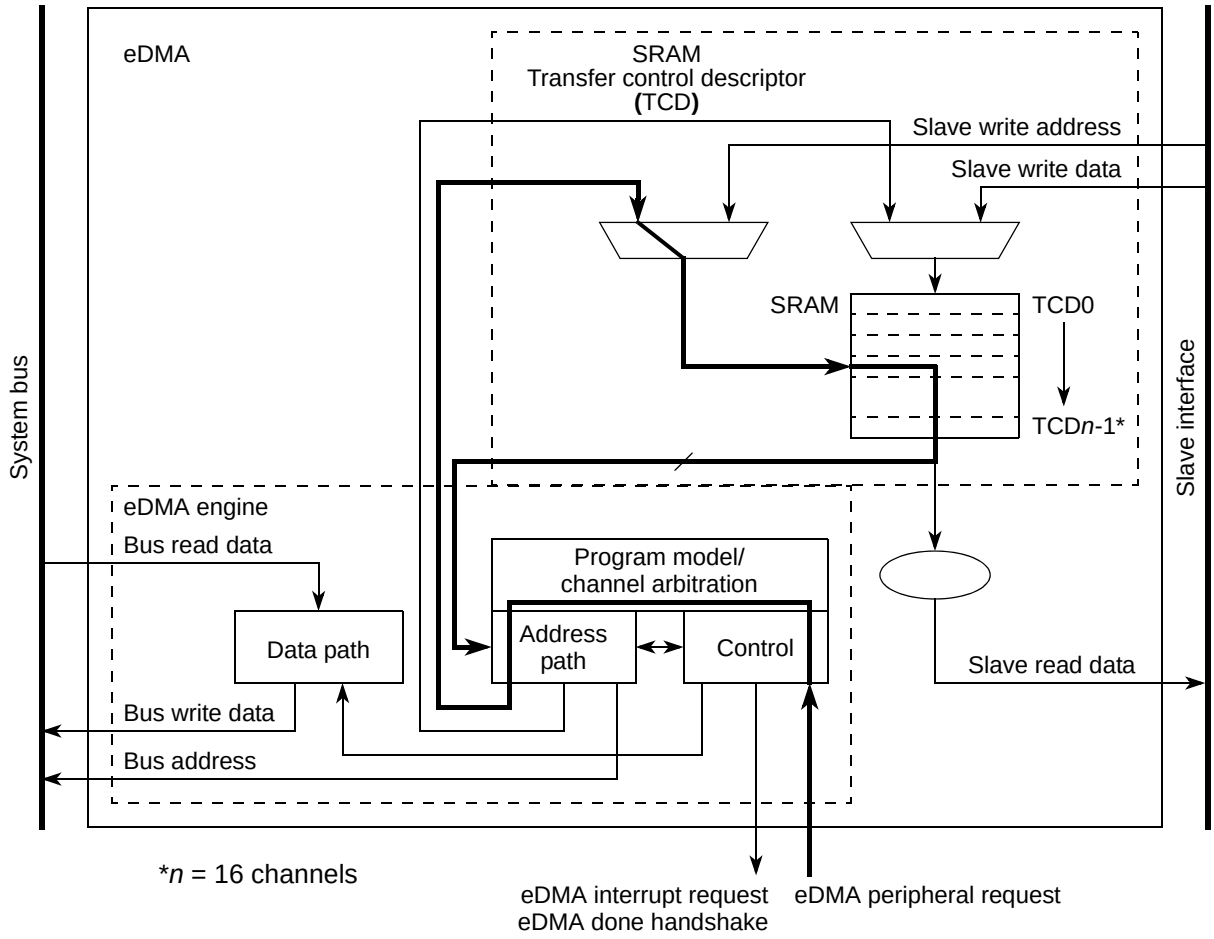


Figure 157. eDMA Operation, Part 1

In the second part of the basic data flow as shown in *Figure 158*, the modules associated with the data transfer (address path, data path, and control) sequence through the required

source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.

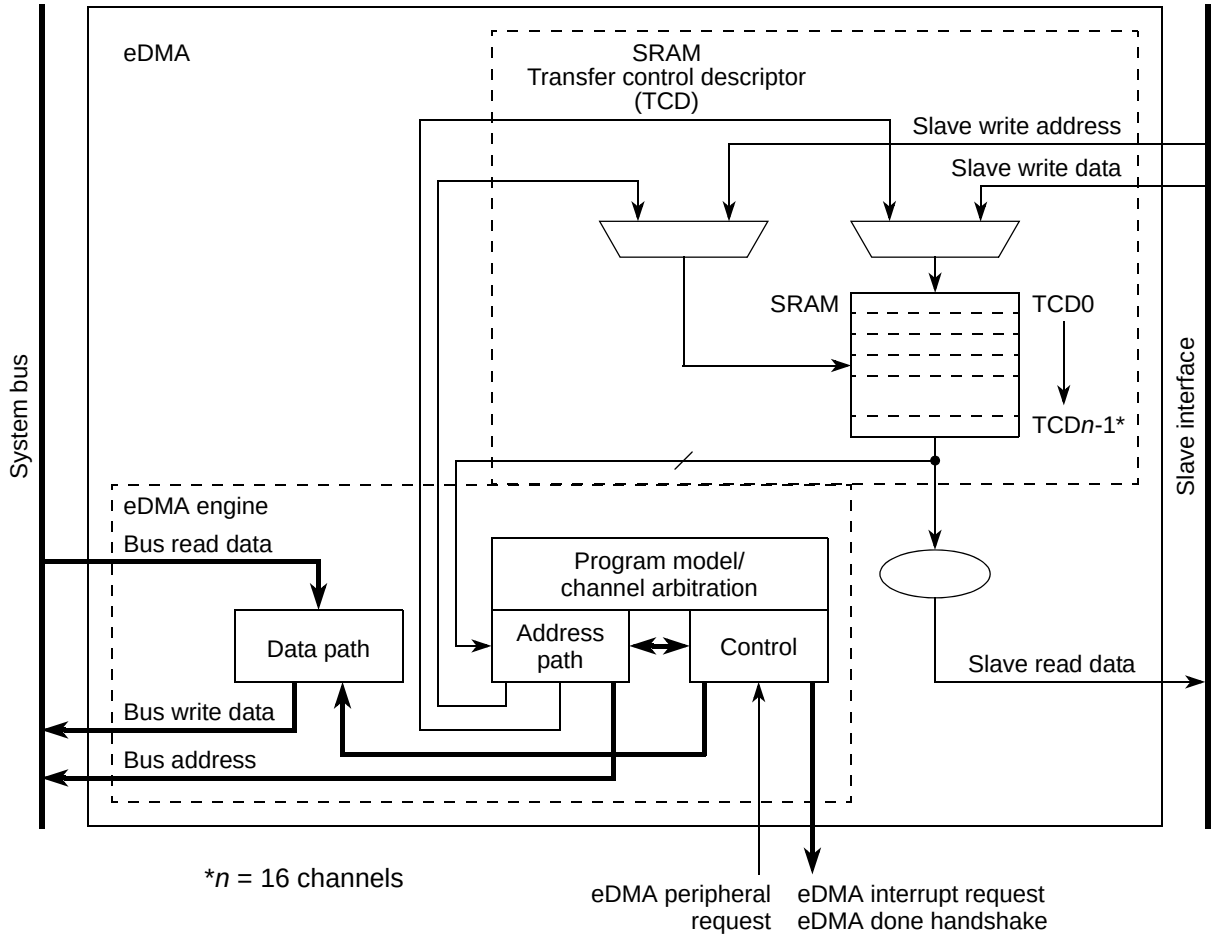


Figure 158. eDMA Operation, Part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in *Figure 159*.

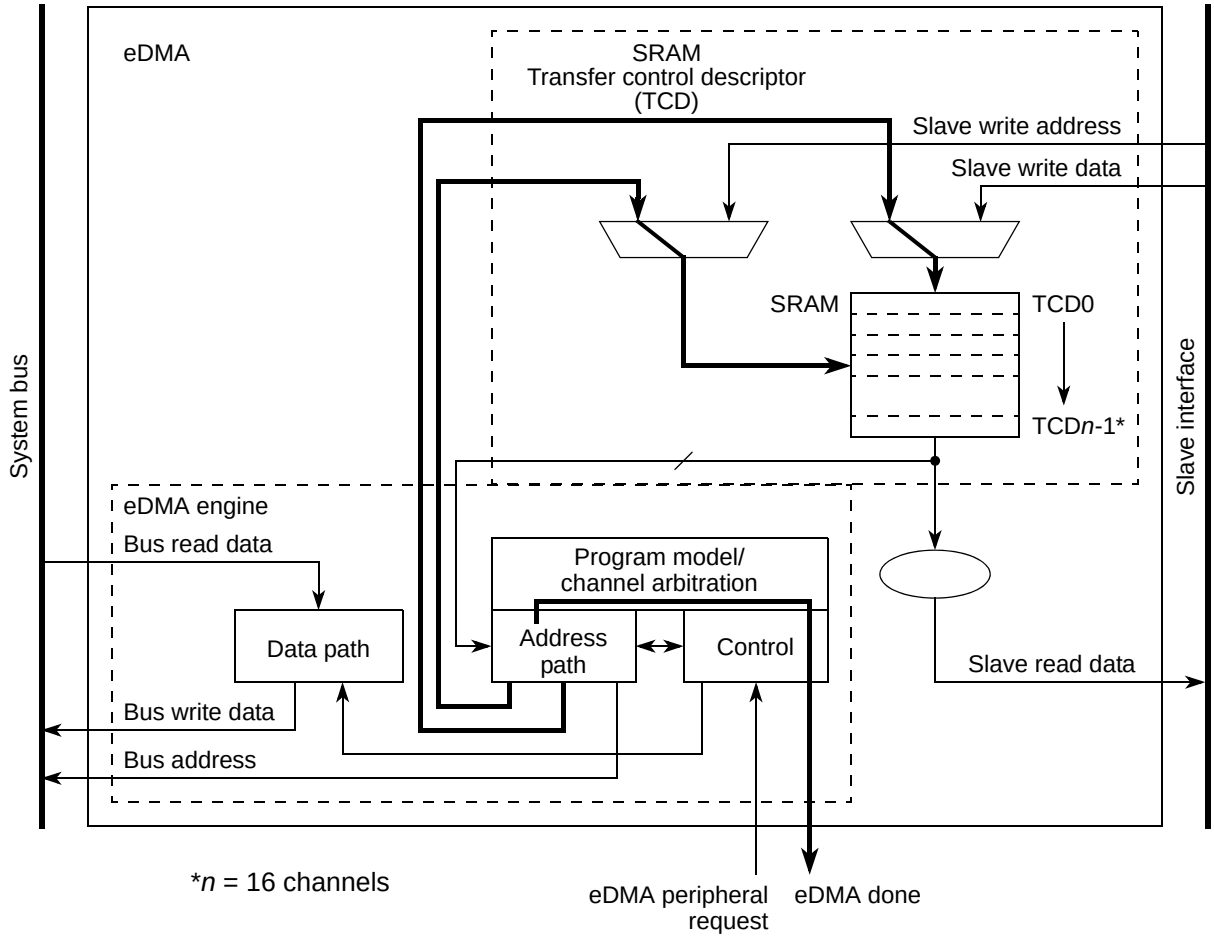


Figure 159. eDMA Operation, Part 3

17.5.2 eDMA performance

This section addresses the performance of the eDMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the eDMA. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests which can be serviced in a fixed time is a more interesting metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the eDMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in *Table 156*. The following assumptions apply to *Table 156* and *Table 157*:

- Platform SRAM can be accessed with zero wait-states when viewed from the AMBA-AHB data phase
- All IPS reads require two wait-states, and IPS writes three wait-states, again viewed from the system bus data phase
- All IPS accesses are 32 bits in size

Table 156 presents a peak transfer rate comparison, measured in megabytes per second. In this table, the Platform_SRAM-to-Platform_SRAM transfers occur at the native platform datapath width, i.e., either 32- or 64-bits per access. For all transfers involving the IPS bus, 32-bit transfer sizes are used. In all cases, the transfer rate includes the time to read the source plus the time to write the destination.

Table 156. eDMA peak transfer rates [MB/s]

Platform Speed, Width	Platform SRAM-to-Platform SRAM	32-bit IPS-to-Platform SRAM	Platform SRAM-to-32-bit IPS
66.7 MHz, 32-bit	133.3	66.7	53.3
66.7 MHz, 64-bit	266.7	66.6	53.3
83.3 MHz, 32-bit	166.7	83.3	66.7
83.3 MHz, 64-bit	333.3	83.3	66.7
100.0 MHz, 32-bit	200.0	100.0	80.0
100.0 MHz, 64-bit	400.0	100.0	80.0
133.3 MHz, 32-bit	266.7	133.3	106.7
133.3 MHz, 64-bit	533.3	133.3	106.7
150.0 MHz, 32-bit	300.0	150.0	120.0
150.0 MHz, 64-bit	600.0	150.0	120.0

The second performance metric is a measure of the number of eDMA requests which can be serviced in a given amount of time. For this metric, it is assumed the peripheral request causes the channel to move a single IPS-mapped operand to/from the platform SRAM. The same timing assumptions used in the previous example apply to this calculation. In particular, this metric also reflects the time required to activate the channel. The eDMA design supports the following hardware service request sequence:

- Cycle 1: ipd_req[n] is asserted
- Cycle 2: The ipd_req[n] is registered locally in the eDMA module and qualified (TCD.start bit initiated requests start at this point with the registering of the IPS write to TCD word7)
- Cycle 3: Channel arbitration begins
- Cycle 4: Channel arbitration completes. The transfer control descriptor local memory read is initiated.
- Cycle 5 - 6: The first two parts of the activated channel's TCD is read from the local memory. The memory width to the eDMA engine is 64 bits, so the entire descriptor can be accessed in four cycles.
- Cycle 7: The first AMBA-AHB read cycle is initiated, as the third part of the channel's TCD is read from the local memory. Depending on the state of the platform's crossbar switch, arbitration at the system bus may insert an additional cycle of delay here.
- Cycle 8 - ?: The last part of the TCD is read in. This cycle represents the 1st data phase for the read, and the address phase for the destination write.
  The exact timing from this point is a function of the response times for the channel's

read and write accesses. In this case of an IPS read and a platform SRAM write, the combined data phase time is 4 cycles. For an SRAM read and IPS write, it is 5 cycles.

- Cycle ?+1: This cycle represents the data phase of the last destination write
- Cycle ?+2: The eDMA engine completes the execution of the inner minor loop and prepares to write back the required TCDn fields into the local memory. TCD word7 is read and checked for channel linking or scatter/gather requests.
- Cycle ?+3: The appropriate fields in the first part of the TCDn are written back into the local memory
- Cycle ?+4: The fields in the second part of the TCDn are written back into the local memory. This cycle coincides with the next channel arbitration cycle start.
- Cycle ?+5: The next channel to be activated performs the read of the first part of its TCD from the local memory. This is equivalent to Cycle 4 for the first channel's service request.

Assuming zero wait states on the AHB system bus, eDMA requests can be processed every 9 cycles. Assuming an average of the access times associated with IPS-to-SRAM (4 cycles) and SRAM-to-IPS (5 cycles), eDMA requests can be processed every 11.5 cycles (4 + (4+5)÷2 + 3). This is the time from Cycle 4 to Cycle "?+5". The resulting peak request rate, as a function of the platform frequency, is shown in *Table 157*. This metric represents millions of requests per second.

Table 157. eDMA peak request rate [MReq/sec]

Platform Speed	Request Rate (zero wait state)	Request Rate (with wait states)
66.6 MHz	7.4	5.8
83.3 MHz	9.2	7.2
100.0 MHz	11.1	8.7
133.3 MHz	14.8	11.6
150.0 MHz	16.6	13.0

A general formula to compute the peak request rate (with overlapping requests) is:

PEAKreq = freq ÷ [entry + (1 + read_ws) + (1 + write_ws) + exit]

where:

PEAKreq - peak request rate

freq - platform frequency

entry - channel startup (4 cycles)

read_ws - wait states seen during the system bus read data phase

write_ws - wait states seen during the system bus write data phase

exit - channel shutdown (3 cycles)

For example: consider a platform with the following characteristics:

- Platform SRAM can be accessed with one wait-state when viewed from the AMBA-AHB data phase
- All IPS reads require two wait-states, and IPS writes three wait-states, again viewed from the system bus data phase
- Platform operates at 150 MHz

For an SRAM to IPS transfer:

PEAKreq = 150 MHz ÷ [4 + (1 + 1) + (1 + 3) + 3] cycles = 11.5 Mreq/sec

For an IPS to SRAM transfer:

PEAKreq = 150 MHz ÷ [4 + (1 + 2) + (1 + 1) + 3] cycles = 12.5 Mreq/sec

Assuming an even distribution of the two transfer types, the average Peak Request Rate would be:

PEAKreq = (11.5 Mreq/sec + 12.5 Mreq/sec) ÷ 2 = 12.0 Mreq/sec

The minimum number of cycles to perform a single read/write, zero wait states on the system bus, from a cold start (where no channel is executing, eDMA is idle) are:

- 11 cycles for a software (TCD.start bit) request
- 12 cycles for a hardware (ipd_req signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the ipd_req signals. For the peak request rate calculations above, the arbitration and request registering is absorbed in or overlap the previous executing channel.

Note: When channel linking or scatter/gather is enabled, a two cycle delay is imposed on the next channel selection and startup. This allows the link channel or the scatter/gather channel to be eligible and considered in the arbitration pool for next channel selection.

17.6 Initialization / Application Information

17.6.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by software (setting the TCD.START bit) or by hardware (slave device asserting its DMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine will read the entire TCD, including the primary transfer control parameter shown in *Table 158*, for the selected channel into its internal address path module. As the TCD is being read, the first

transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the DMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

Table 158. TCD Primary Control and Status Fields

TCD Field Name	Description
START	Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 160 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

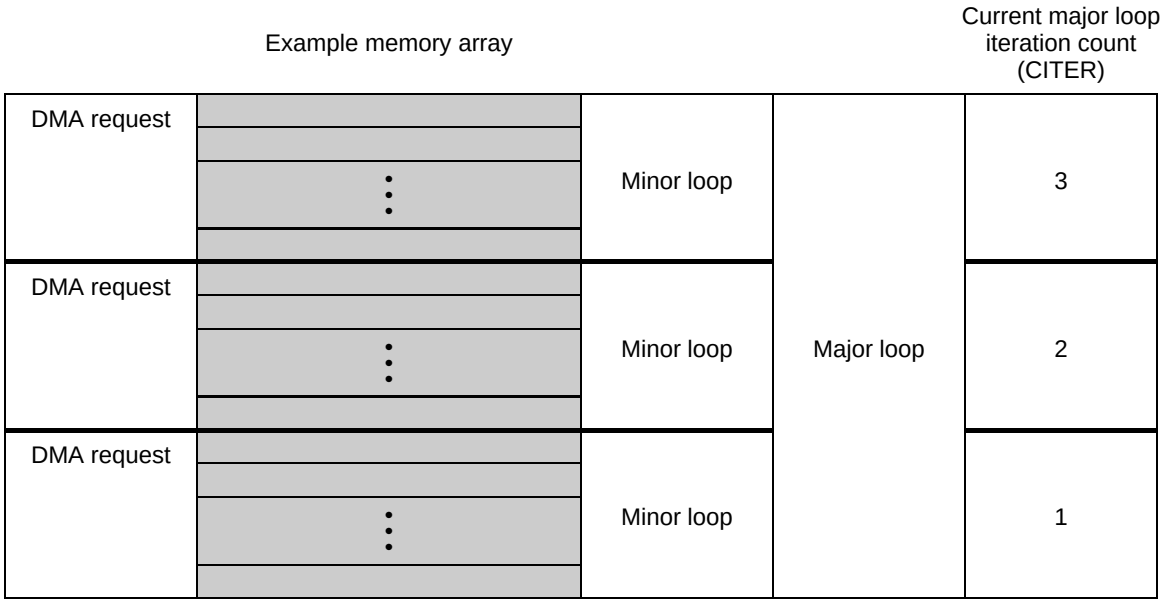


Figure 160. Example of Multiple Loop Iterations

Figure 161 lists the memory array terms and how the TCD settings interrelate.

xADDR: (Starting address)	xSIZE: (Size of one data transfer)	Minor loop (NBYTES in minor loop, often the same value as xSIZE)	Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE) Each DMA source (S) and destination (D) has its own: • Address (xADDR) • Size (xSIZE) • Offset (xOFF) • Modulo (xMOD) • Last address adjustment (xLAST) where x = S or D Peripheral queues typically have size and offset equal to NBYTES
• • •	• • •	Minor loop	
xLAST: Number of bytes added to current address after major loop (typically used to loop back)	• • •	Last minor loop	

Figure 161. Memory array terms

17.6.2 DMA programming errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of two errors: group-priority error and channel-priority error, or EDMA_ESR[GPE] and EDMA_ESR[CPE], respectively.

For all error types other than group- or channel-priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

Channel-priority errors are identified within a group after that group has been selected as the active group. For the example, all of the channel priorities in group 1 are unique, but some of the channel priorities in group 0 are the same:

1. The DMA is configured for fixed-group and fixed-channel arbitration modes.
2. Group 1 is the highest priority and all channels are unique in that group.
3. Group 0 is the next highest priority and has two channels with the same priority level.
4. If group 1 has any service requests, those requests will be executed.
5. After all of group 1 requests have completed, group 0 will be the next active group.
6. If group 0 has a service request, then an undefined channel in group 0 will be selected and a channel-priority error will occur.
7. This will repeat until the all of group 0 requests have been removed or a higher priority group 1 request comes in.

In this sequence, for item 2, the DMA acknowledge lines will assert only if the selected channel is requesting service via the DMA peripheral request signal. If interrupts are enabled for all channels, the user will receive an error interrupt, but the channel number for the EDMA_ER and the error interrupt request line are undetermined because they reflect the undefined channel. A group-priority error is global and any request in any group will cause a group-priority error.

If priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

17.6.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in *Table 159*. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 159. DMA Request Summary for eDMA

DMA Request	Channel	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source

Table 159. DMA Request Summary for eDMA (continued)

DMA Request	Channel	Source	Description
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source
DMA_MUX_CHCONFIG12_SOURCE	12	DMA_MUX.CHCONFIG12[SOURCE]	DMA MUX channel 12 source
DMA_MUX_CHCONFIG13_SOURCE	13	DMA_MUX.CHCONFIG13[SOURCE]	DMA MUX channel 13 source
DMA_MUX_CHCONFIG14_SOURCE	14	DMA_MUX.CHCONFIG14[SOURCE]	DMA MUX channel 14 source
DMA_MUX_CHCONFIG15_SOURCE	15	DMA_MUX.CHCONFIG15[SOURCE]	DMA MUX channel 15 source

17.6.4 DMA Arbitration Mode Considerations

17.6.4.1 Fixed-Group Arbitration, Fixed-Channel Arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the eDMA is programmed so the channels within one group use fixed priorities, and that group is assigned the highest fixed priority of all groups, it is possible for that group to take all the bandwidth of the eDMA controller. That is, no other groups will be serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

17.6.4.2 Round-Robin Group Arbitration, Fixed-Channel Arbitration

The occurrence of one or more DMA requests from one or more groups, the channel with the highest priority from a specific group will be serviced first. Groups are serviced starting with the highest group number with a service request and rotating through to the lowest group number containing a service request.

After the channel request is serviced, the group round robin algorithm will select the highest pending request from the next group in the round-robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence, or skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round robin service rate, then that channel will always be serviced before lower priority channels in the same group, and the lower priority channels will never be serviced. The advantage of this scenario is that no one group will consume all the eDMA bandwidth. The highest priority channel selection

latency is potentially greater than fixed/fixed arbitration. Excessive request rates on high-priority channels can prevent the servicing of lower priority channels in the same group.

17.6.4.3 Round-Robin Group Arbitration, Round-Robin Channel Arbitration

Groups will be serviced as described in *Section 17.6.4.2: Round-Robin Group Arbitration, Fixed-Channel Arbitration*, but this time channels will be serviced in channel number order. One channel only is serviced from each requesting group for each round robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round-robin manner, any channel that generates DMA requests faster than a combination of the group round-robin service rate and the channel service rate for its group will not prevent the servicing of other channels in its group. Any DMA requests that are not serviced are simply lost, but at least one channel will be serviced.

This scenario ensures that all channels will be guaranteed service at some point, regardless of the request rates. However, the potential latency could be high. All channels are treated equally. Priority levels are not used in round-robin/round-robin mode.

17.6.4.4 Fixed-Group Arbitration, Round-Robin Channel Arbitration

The highest priority group with a request will be serviced. Lower priority groups will be serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in *Section 17.6.4.1: Fixed-Group Arbitration, Fixed-Channel Arbitration*, but all the channels in the highest priority group will get serviced. Service latency will be short on the highest priority group, but could potentially get longer and longer as the group priority decreases.

17.6.5 DMA transfer

17.6.5.1 Single request

To perform a simple transfer of *n* bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer will begin after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD.DONE bit will be set and an interrupt will be generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to

match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

TCD.CITER = TCD.BITER = 1

TCD.NBYTES = 16

TCD.SADDR = 0x1000

TCD.SOFF = 1

TCD.SSIZE = 0

TCD.SLAST = –16

TCD.DADDR = 0x2000

TCD.DOFF = 4

TCD.DSIZE = 2

TCD.DLAST_SGA= -16

TCD.INT_MAJ = 1

TCD.START = 1 (Must be written last after all other fields have been initialized)

All other TCD fields = 0

This would generate the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word(0x2000) → first iteration of the minor loop
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word(0x2004) → second iteration of the minor loop
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
   f) write_word(0x2008) → third iteration of the minor loop
   g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
   h) write_word(0x200c) → last iteration of the minor loop → major loop complete
6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

17.6.5.2 Multiple requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA_ERQR,

channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that TCD.START = 0.

TCD.CITER = TCD.BITER = 2

TCD.NBYTES = 16

TCD.SADDR = 0x1000

TCD.SOFF = 1

TCD.SSIZE = 0

TCD.SLAST = –32

TCD.DADDR = 0x2000

TCD.DOFF = 4

TCD.DSIZE = 2

TCD.DLAST_SGA= –32

TCD.INT_MAJ = 1

TCD.START = 0 (Must be written last after all other fields have been initialized)

All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word(0x2000) → first iteration of the minor loop
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word(0x2004) → second iteration of the minor loop
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
   f) write_word(0x2008) → third iteration of the minor loop
   g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
   h) write_word(0x200c) → last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
7. eDMA engine writes: TCD.ACTIVE = 0.
8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
    a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
    b) write_word(0x2010) → first iteration of the minor loop
    c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
    d) write_word(0x2014) → second iteration of the minor loop
    e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101a), read_byte(0x101b)
    f) write_word(0x2018) → third iteration of the minor loop
    g) read_byte(0x101c), read_byte(0x101d), read_byte(0x101e), read_byte(0x101f)
    h) write_word(0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

17.6.5.3 Modulo feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit bitfield for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 160 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 160. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

17.6.6 TCD status

17.6.6.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method may be extracted from the sequence below. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a 1. Polling the TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (channel service request via software).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution, regardless of how the channel was activated.

17.6.6.2 Active channel TCD Reads

The eDMA will read back the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

17.6.6.3 Preemption status

Preemption is available only when fixed arbitration is selected for both group- and channel-arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed-channel arbitration mode, the determination of the relative

priority of the actively running and the outstanding requests become undefined. Channel and group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

17.6.7 Channel linking

Channel linking (or chaining) is a mechanism in which one channel sets the TCD.START bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

TCD.CITER.E_LINK = 1

TCD.CITER.LINKCH = 0xC

TCD.CITER value = 0x4

TCD.MAJOR.E_LINK = 1

TCD.MAJOR.LINKCH = 0x7

will execute as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit
3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

Note: After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be equal or a configuration error will be reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.

Table 161 summarizes how a DMA channel can link to another DMA channel, i.e, use another channel's TCD, at the end of a loop.

Table 161. Channel Linking Parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of minor loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration).
	citer.linkch	Link channel number when linking at end of minor loop (current iteration).
Link at end of major loop	major.e_link	Enable channel-to-channel linking on major loop completion.
	major.linkch	Link channel number when linking at end of major loop.

17.6.8 Dynamic programming

17.6.8.1 Dynamic channel linking

Dynamic channel linking is the process of setting the TCD.major.e_link bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

A coherency model is needed because the user is allowed to change the configuration during execution. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e_link bit at the same time the eDMA engine is retiring the channel. The TCD.major.e_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The coherency model in *Table 162* is recommended when executing a dynamic channel link request.

Table 162. Coherency model for a dynamic channel link request

Step	Action
1	Write 1b to the TCD.major.e_link bit.
2	Read back the TCD.major.e_link bit.
3	Test the TCD.major.e_link request status: – If TCD.major.e_link = 1b, the dynamic link attempt was successful. – If TCD.major.e_link = 0b, the attempted dynamic link did not succeed (the channel was already retiring).

For this request, the TCD local memory controller forces the TCD.major.e_link bit to zero on any writes to a channel's TCD.word7 after that channel's TCD.done bit is set, indicating the major loop is complete.

Note: The user must clear the TCD.done bit before writing the TCD.major.e_link bit. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

17.6.8.2 Dynamic scatter/gather

Dynamic scatter/gather is the process of setting the TCD.e_sg bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic scatter/gather operation by enabling the TCD.e_sg bit at the same time the eDMA engine is retiring the channel. The TCD.e_sg would be set in the programmer's model, but it would be unclear whether the actual scatter/gather request was honored before the channel retired.

Two methods for this coherency model are shown in the following subsections. Method 1 has the advantage of reading the major.linkch field and the e_sg bit with a single read. For both dynamic channel linking and scatter/gather requests, the TCD local memory controller forces the TCD.major.e_link and TCD.e_sg bits to zero on any writes to a channel's TCD.word7 if that channel's TCD.done bit is set indicating the major loop is complete.

Note: The user must clear the TCD.done bit before writing the TCD.major.e_link or TCD.e_sg bits. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

17.6.8.2.1 Method 1 (channel not using major loop channel linking)

For a channel not using major loop channel linking, the coherency model in *Table 163* may be used for a dynamic scatter/gather request.

When the TCD.major.e_link bit is zero, the TCD.major.linkch field is not used by the eDMA. In this case, the TCD.major.linkch bits may be used for other purposes. This method uses the TCD.major.linkch field as a TCD identification (ID).

Table 163. Coherency model for method 1

Step	Action
1	When the descriptors are built, write a unique TCD ID in the TCD.major.linkch field for each TCD associated with a channel using dynamic scatter/gather.
2	Write 1b to theTCD.d_req bit. **Note:** Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offset value.
3	Write theTCD.dlast_sga field with the scatter/gather address.
4	Write 1b to the TCD.e_sg bit.
5	Read back the 16 bit TCD control/status field.
6	Test the TCD.e_sg request status and TCD.major.linkch value: – If e_sg = 1b, the dynamic link attempt was successful. – If e_sg = 0b and the major.linkch (ID) did not change, the attempted dynamic link did not succeed (the channel was already retiring). – If e_sg = 0b and the major.linkch (ID) changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).

17.6.8.2.2 Method 2 (channel using major loop linking)

For a channel using major loop channel linking, the coherency model in *Table 164* may be used for a dynamic scatter/gather request. This method uses the TCD.dlast_sga field as a TCD identification (ID).

Table 164. Coherency model for method 2

Step	Action
1	Write 1b to theTCD.d_req bit. **Note:** Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offset value.
2	Write theTCD.dlast_sga field with the scatter/gather address.
3	Write 1b to the TCD.e_sg bit.
4	Read back the TCD.e_sg bit.
5	Test the TCD.e_sg request status: – If e_sg = 1b, the dynamic link attempt was successful. – If e_sg = 0b, read the 32 bit TCD dlast_sga field. – If e_sg = 0b and the dlast_sga did not change, the attempted dynamic link did not succeed (the channel was already retiring). – If e_sg = 0b and the dlast_sga changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).

18 eDMA Channel Multiplexer (DMA_MUX)

18.1 Introduction

The DMA Mux allows to route 56 DMA sources (called slots) to 32 eDMA channels. This is illustrated in *Figure 162*.

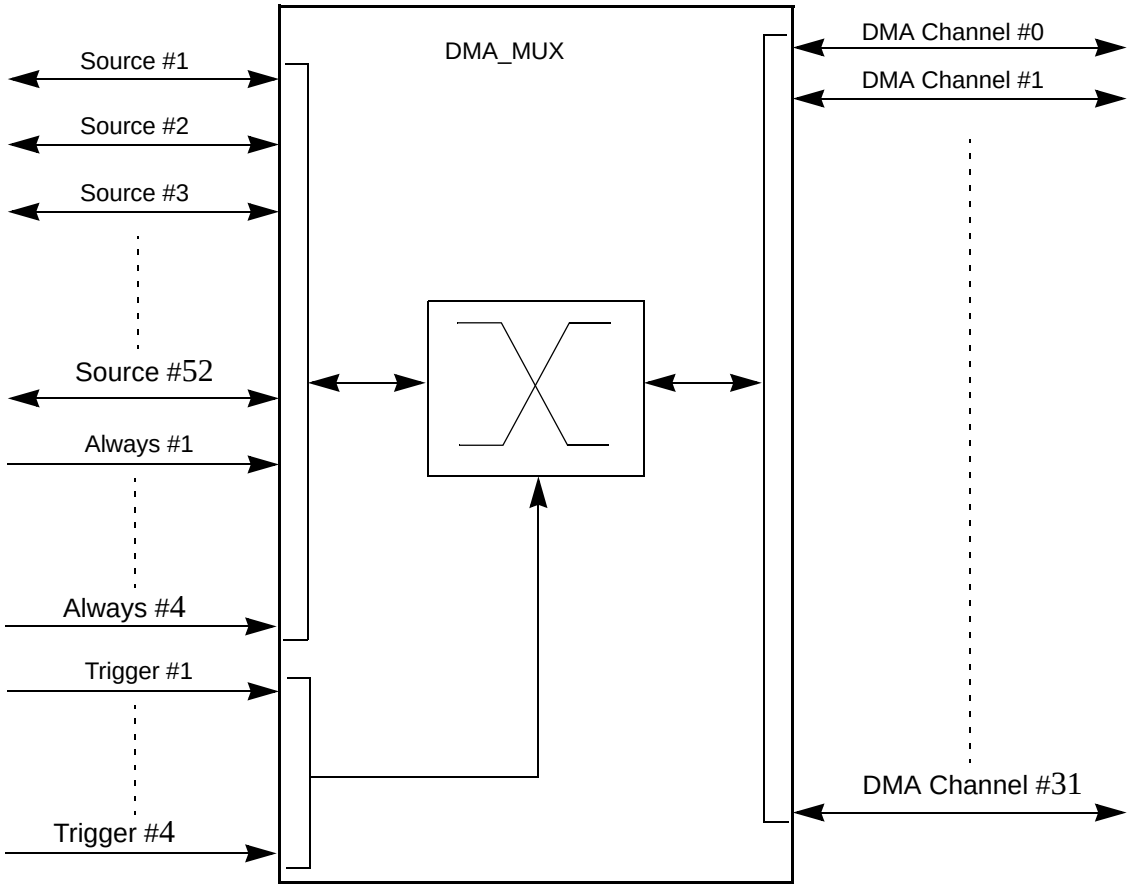


Figure 162. DMA_MUX block diagram

18.2 Features

The eDMA Channel Mux provides these features:

- 52 peripheral slots + 4 always-on slots can be routed to 32 channels
- 64 independently selectable eDMA channels routers
  - the first 4 channels additionally provide a trigger functionality
- Each channel router can be assigned to one of 52 possible peripheral eDMA slots or to one of the 4 always-on slots.

18.2.1 Modes of operation

The following operation modes are available:

- Disabled mode

  In this mode, the eDMA channel is disabled. Since disabling and enabling of eDMA channels is done primarily via the eDMA configuration registers, this mode is used mainly as the reset state for a eDMA channel in the eDMA Channel Mux. It may also be used to temporarily suspend a eDMA channel while reconfiguration of the system takes place (e.g. changing the period of a eDMA trigger).

- Normal mode

  In this mode, a eDMA source (such as DSPI transmit or DSPI receive for example) is routed directly to the specified eDMA channel. The operation of the eDMA Mux in this mode is completely transparent to the system.

- Periodic Trigger mode

  In this mode, a eDMA source may only request a eDMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the PIT_RTI. This mode is only available for channels 0-3.

18.3 External signal description

18.3.1 Overview

The eDMA Mux has no external pins.

18.4 Memory map and register definition

This section provides a detailed description of all memory-mapped registers in the eDMA Mux.

Table 165 shows the memory map for the eDMA Mux. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the eDMA Mux.

Table 165. DMA_MUX memory map

Base address: 0xFFFD_C000		
Address offset	**Register Name**	**Location**
0x00	Channel #0 Configuration (CHCONFIG0)	*on page 414*
0x01	Channel #1 Configuration (CHCONFIG1)	*on page 414*
..	..	
0x1F	Channel #*31* Configuration (CHCONFIG*31*)	*on page 414*

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit

READ/WRITE to address ‘Base + 0x00’, but performing a 32-bit access to address ‘Base + 0x01’ is illegal.

18.4.1 Register descriptions

The following memory-mapped registers are available in the eDMA Channel Mux.

18.4.1.1 Channel Configuration Registers

Each of the eDMA channels can be independently enabled/disabled and associated with one of the eDMA slots (peripheral slots or always-on slots) in the system.

Address: Base + *#n* Access: User read/write

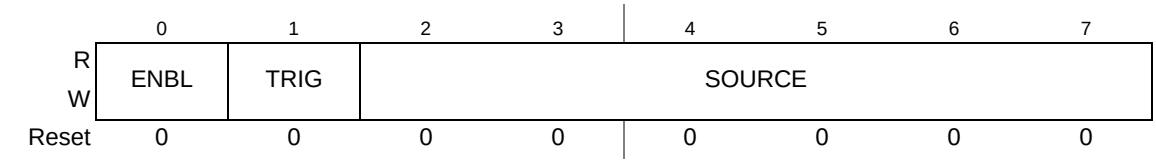


Figure 163. Channel Configuration Registers (CHCONFIG*#n*)

Table 166. CHCONFIG*xx* Field Descriptions

Field	Description
7 ENBL	eDMA Channel Enable. ENBL enables the eDMA Channel 0 eDMA channel is disabled. This mode is primarily used during configuration of the eDMA Mux. The eDMA has separate channel enables/disables, which should be used to disable or re-configure a eDMA channel. 1 eDMA channel is enabled
6 TRIG	eDMA Channel Trigger Enable (for triggered channels only). TRIG enables the periodic trigger capability for the eDMA Channel 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the eDMA Channel will simply route the specified source to the eDMA channel. 1 Triggering is enabled
5–0 SOURCE	eDMA Channel Source (slot). SOURCE specifies which eDMA source, if any, is routed to a particular eDMA channel. For further details about the peripherals and their slot numbers, refer to *Table 168*.

Table 167. Channel and Trigger Enabling

ENBL	TRIG	Function	Mode
0	X	eDMA Channel is disabled	Disabled Mode
1	0	eDMA Channel is enabled with no triggering (transparent)	Normal Mode
1	1	eDMA Channel is enabled with triggering	Periodic Trigger Mode

Note: Setting multiple CHCONFIG registers with the same Source value will result in unpredictable behavior.

Note: Before changing the trigger or source settings a eDMA channel must be disabled via the CHCONFIG[#n].ENBL bit.

18.4.2 DMA_MUX inputs

18.4.2.1 DMA_MUX peripheral sources

Table 168. DMA channel mapping

DMA_MUX channel	Module	DMA requesting module
0	—	Always disabled
1	DSPI 0	DSPI_0 TX
2	DSPI 0	DSPI_0 RX
3	DSPI 1	DSPI_1 TX
4	DSPI 1	DSPI_1 RX
5	DSPI 2	DSPI_2 TX
6	DSPI 2	DSPI_2 RX
7	DSPI 3	DSPI_3 TX
8	DSPI 3	DSPI_3 RX
9	DSPI 4	DSPI_4 TX
10	DSPI 4	DSPI_4 RX
11	DSPI 5	DSPI_5 TX
12	DSPI 5	DSPI_5 RX
13	DSPI 6	DSPI_6 RX
14	DSPI 6	DSPI_6 RX
15	DSPI 7	DSPI_7 RX
16	DSPI 7	DSPI_7 RX
17	eMIOS 0	EMIOS0_CH0
18	eMIOS 0	EMIOS0_CH1
19	eMIOS 0	EMIOS0_CH9
20	eMIOS 0	EMIOS0_CH18
21	eMIOS 0	EMIOS0_CH25
22	eMIOS 0	EMIOS0_CH26
23	eMIOS 1	EMIOS1_CH0
24	eMIOS 1	EMIOS1_CH9
25	eMIOS 1	EMIOS1_CH17
26	eMIOS 1	EMIOS1_CH18
27	eMIOS 1	EMIOS1_CH25
28	eMIOS 1	EMIOS1_CH26
29	ADC 0	ADC0_EOC
30	ADC 1	ADC1_EOC

Table 168. DMA channel mapping (continued)

DMA_MUX channel	Module	DMA requesting module
31	I^2C	IIC_RX
32	I^2C	IIC_TX
33	LINFlexD_0	LINFlexD_0_RX
34	LINFlexD_0	LINFlexD_0_TX
35	LINFlexD_1	LINFlexD_1_RX
36	LINFlexD_1	LINFlexD_1_TX
37	LINFlexD_2	LINFlexD_2_RX
38	LINFlexD_2	LINFlexD_2_TX
39	LINFlexD_3	LINFlexD_3_RX
40	LINFlexD_3	LINFlexD_3_TX
41	LINFlexD_4	LINFlexD_4_RX
42	LINFlexD_4	LINFlexD_4_TX
43	LINFlexD_5	LINFlexD_5_RX
44	LINFlexD_5	LINFlexD_5_TX
45	LINFlexD_6	LINFlexD_6_RX
46	LINFlexD_6	LINFlexD_6_TX
47	LINFlexD_7	LINFlexD_7_RX
48	LINFlexD_7	LINFlexD_7_TX
49	LINFlexD_8	LINFlexD_8_RX
50	LINFlexD_8	LINFlexD_8_TX
51	LINFlexD_9	LINFlexD_9_RX
52	LINFlexD_9	LINFlexD_9_TX
53	—	—
54	—	—
55	—	—
56	—	—
57	—	—
58	—	—
59	—	—
60	—	ALWAYS ENABLED
61	—	ALWAYS ENABLED
62	—	ALWAYS ENABLED
63	—	ALWAYS ENABLED

18.4.2.2 DMA_MUX periodic trigger inputs

Table 169. DMA_MUX periodic trigger inputs

DMA_MUX trigger input	PIT_RTI channel
Trigger #1	PIT0
Trigger #2	PIT1
Trigger #3	PIT4
Trigger #4	PIT5

18.5 Functional description

This section provides a functional description of the DMA Mux. The primary purpose of the DMA Mux is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA Mux is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in *Section 18.6.2: Enabling and Configuring Sources* is followed, the configuration of the DMA MUX may be changed during the normal operation of the system.

Functionally, the DMA Mux channels may be divided into two classes: Channels, which implement the normal routing functionality plus periodic triggering capability, and channels, which implement only the normal routing functionality.

18.5.1 DMA Channels with periodic triggering capability

Besides the normal routing functionality, the first 4 channels (CH0–CH3) of the DMA Mux provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the PIT_RTI; as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT_RTI. For details, refer to *Section 31.4: Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI)*.

Note: Because of the dynamic nature of the system (i.e. DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

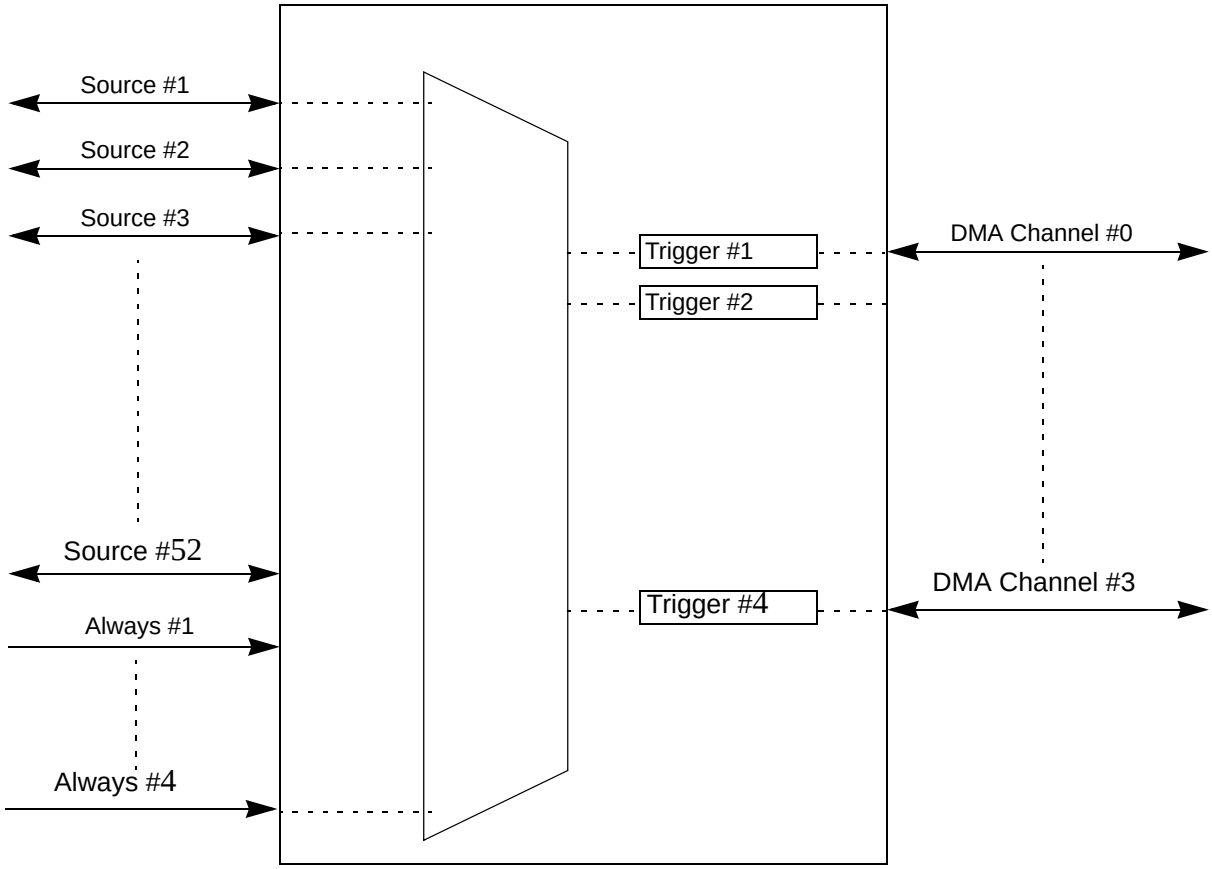


Figure 164. DMA Mux triggered channels

The DMA channel triggering capability allows the system to “schedule” regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the Peripheral to the DMA until a trigger event has been seen. This is illustrated in *Figure 165*.

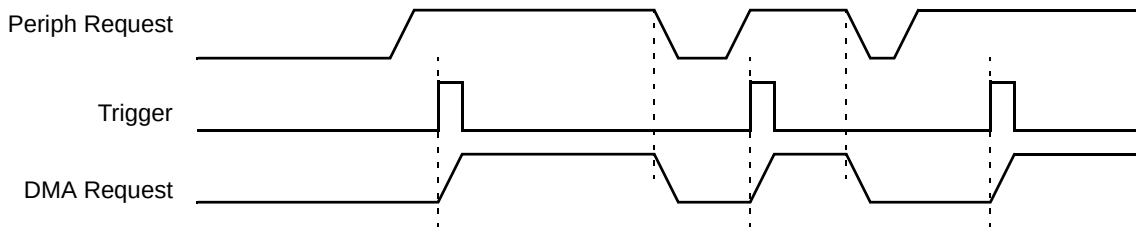


Figure 165. DMA Mux Channel Triggering: Normal Operation

Once the DMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in *Figure 166*.

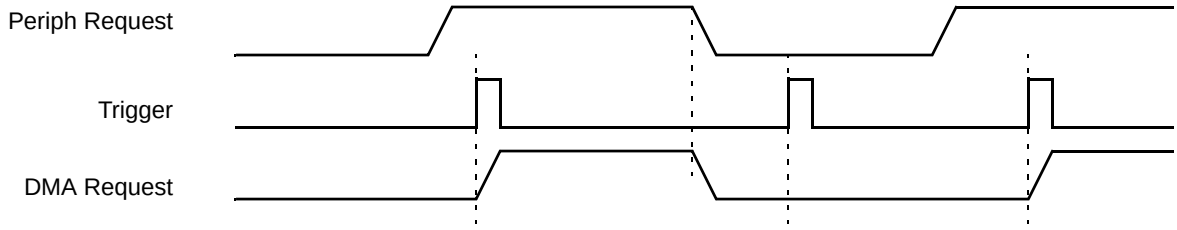


Figure 166. DMA Mux Channel Triggering: Ignored Trigger

This triggering capability may be used with any peripheral that supports DMA transfers, and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. Once setup, the SPI will request DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5µs (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO Ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (i.e.-resolution, range of values, etc.) may be found in the *Section 31.4: Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI)*.

18.5.2 DMA Channels with no triggering capability

The other channels of the DMA Mux provide the normal routing functionality as described in *Section 18.2.1: Modes of operation*.

18.5.3 "Always Enabled" DMA Sources

In addition to the peripherals that can be used as DMA sources, there are 4 additional DMA sources that are "always enabled". Unlike the peripheral DMA sources, where the peripheral controls the flow of data during DMA transfers, the "always enabled" sources provide no such "throttling" of the data transfers. These sources are most useful in the following cases:

- Doing DMA transfers to/from GPIO - Moving data from/to one or more GPIO pins, either un-throttled (i.e.-as fast as possible), or periodically (using the DMA triggering capability).
- Doing DMA transfers from memory to memory - Moving data from memory to memory, typically as fast as possible, sometimes with software activation.
- Doing DMA transfers from memory to the external bus (or vice-versa) - Similar to memory to memory transfers, this is typically done as quickly as possible.
- Any DMA transfer that requires software activation - Any DMA transfer that should be explicitly started by software.

In cases where software should initiate the start of a DMA transfer, a "always enabled" DMA source can be used to provide maximum flexibility. When activating a DMA channel via software, subsequent executions of the minor loop require a new "start" event be sent. This can either be a new software activation, or a transfer request from the DMA Channel Mux. The options for doing this are:

- Transfer all data in a single minor loop. By configuring the DMA to transfer all of the data in a single minor loop (i.e.-major loop counter = 1), no re-activation of the channel is necessary. The disadvantage to this option is the reduced granularity in determining the load that the DMA transfer will incur on the system. For this option, the DMA channel should be disabled in the DMA Channel Mux.
- Use explicit software re-activation. In this option, the DMA is configured to transfer the data using both minor and major loops, but the processor is required to re-activate the channel (by writing to the DMA registers) *after every minor loop*. For this option, the DMA channel should be disabled in the DMA Channel Mux.
- Use a "always enabled" DMA source. In this option, the DMA is configured to transfer the data using both minor and major loops, and the DMA Channel Mux does the channel re-activation. For this option, the DMA channel should be enabled and pointing to an "always enabled" source. Note that the re-activation of the channel can be continuous (DMA triggering is disabled) or can use the DMA triggering capability. In this manner, it is possible to execute periodic transfers of packets of data from one source to another, without processor intervention.

18.6 Initialization/Application Information

18.6.1 Reset

The reset state of each individual bit is shown within the Register Description section (See *Section 18.4.1: Register descriptions*). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

18.6.2 Enabling and Configuring Sources

Enabling a source with periodic triggering

1. Determine with which DMA channel the source will be associated. Note that only the first 4 DMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point
4. Configure the corresponding timer
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set

Example 1 Configure source #5 Transmit for use with DMA Channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the DMA, including enabling the channel
3. Configure a timer for the desired trigger interval
4. Write 0xC5 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
        :
        :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

Enabling a source without periodic triggering

1. Determine with which DMA channel the source will be associated. Note that only the first 4 DMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point
4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared

Example 2 Configure source #5 Transmit for use with DMA Channel 2, with no periodic triggering capability.

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the DMA, including enabling the channel
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
        :
        :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

Disabling a source

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

Switching the source of a DMA Channel

1. Disable the DMA channel in the DMA and re-configure the channel for the new source
2. Clear the ENBL and TRIG bits of the DMA channel
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set

Example 3 Switch DMA Channel 8 from source #5 transmit to source #7 transmit

1. In the DMA configuration registers, disable DMA channel 8 and re-configure it to handle the transfers to peripheral slot 7. This example assumes channel 8 doesn't have triggering capability.
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08)
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08). (In this example, setting the TRIG bit would have no effect, due to the assumption that channels 8 does not support the periodic triggering functionality).

The following code example illustrates steps #2 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);
```

```
In File main.c:
#include "registers.h"
        :
        :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

19 Interrupt Controller (INTC)

19.1 Introduction

This chapter describes the interrupts and the interrupt controller (INTC), which schedules interrupt requests (IRQs) from software and internal peripherals to the e200z4d and e200z0h cores. The INTC provides interrupt prioritization and preemption, interrupt masking, interrupt priority elevation, and protocol support. The INTC supports 279 interrupt requests.

The INTC has two independent sets of priority arbitration/comparison, request selection, vector encoder and acknowledge logic—one set for each CPU. This allows each CPU to handle its software-assigned interrupt requests independently of the other CPU's operation, and provides flexibility for the user to decide which core should handle which interrupt sources in the application. This flexibility comes from a set of configuration bits that allows any interrupt source to generate an interrupt request to either the e200z4d or e200z0h or to both the e200z4d and e200z0h cores.

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 settable interrupt requests, i.e., by using application software to assert an interrupt request. These same software settable 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 settable interrupt request to finish the servicing in a lower priority ISR

19.2 Features

- Supports 238 peripherals and 8 software-configurable interrupt request sources
- Each interrupt source can be steered by software to processor 0 (e200z4d), processor 1 (e200z0h), or both processors interrupt request outputs.

Note: By default, processor 0 (e200z4d) receives all interrupt requests, so backward compatibility with single processor systems is maintained.

- 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 170. Interrupt sources available

Interrupt sources (246)	Number available
Software	8
ECSM	1
DMA	34
Software Watchdog (SWT)	1
STM	4
Flash/RAM ECC (SEC-DED)	2
Real Time Counter (RTC/API)	2
System Integration Unit Lite (SIUL)	3
WakeUp Unit (WKPU)	4
MC_ME	4
MC_RGM	1
FXOSC	1
SXOSC	1
Periodic Interrupt Timer (PIT_RTI)	9
Analog to Digital Converter 0 (ADC0)	2
Analog to Digital Converter 1 (ADC1)	2
FlexCAN 0 (CAN0)	8
FlexCAN 1 (CAN1)	8
FlexCAN 2 (CAN2)	8
FlexCAN 3 (CAN3)	8
FlexCAN 4 (CAN4)	8
FlexCAN 5 (CAN5)	8
LINFlexD_0	3
LINFlexD_1	3
LINFlexD_2	3
LINFlexD_ 3	3
LINFlexD_4	3
LINFlexD_5	3
LINFlexD_6	3
LINFlexD_7	3
LINFlexD_8	3
LINFlexD_9	3
DSPI 0	5
DSPI 1	5

Table 170. Interrupt sources available (continued)

Interrupt sources (246)	Number available
DSPI 2	5
DSPI 3	5
DSPI 4	5
DSPI 5	5
DSPI 6	5
DSPI 7	5
Inter-IC Bus Interface Controller (I2C)	1
Enhanced Modular I/O Subsystem 0 (eMIOS0)	16
Enhanced Modular I/O Subsystem 1 (eMIOS1)	16
Ethernet (FEC)	3
FlexRay	10
Semaphore	2
CSE	1

19.3 Block diagram

Figure 167 provides a block diagram of the INTC.

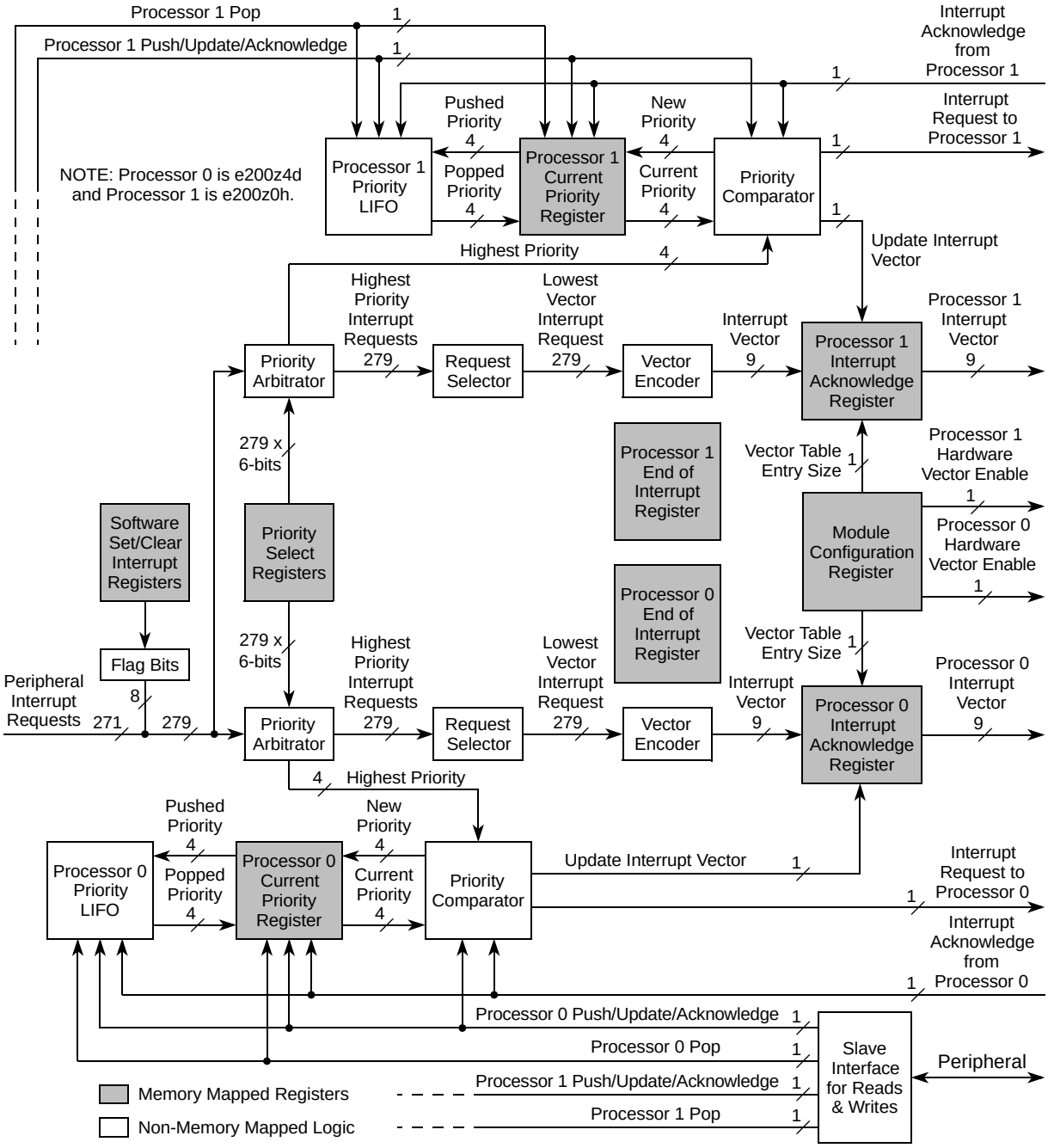


Figure 167. INTC block diagram

19.4 Modes of operation

The interrupt controller has two handshaking modes with the processor: software vector mode and hardware vector mode. The state of the hardware vector enable bit, INTC_MCR[HVEN_PRC*n*], independently determines which mode is used for each CPU.

In debug mode the interrupt controller operation is identical to its normal operation of software vector mode or hardware vector mode.

19.4.1 Software Vector mode

In software vector mode, as shown in *Figure 168*, the CPU branches to a common interrupt exception handler whose location is determined by an address derived from special purpose registers IVPR and IVOR4. The interrupt exception handler reads the INTC_IACKR to determine the vector of the interrupt request source.

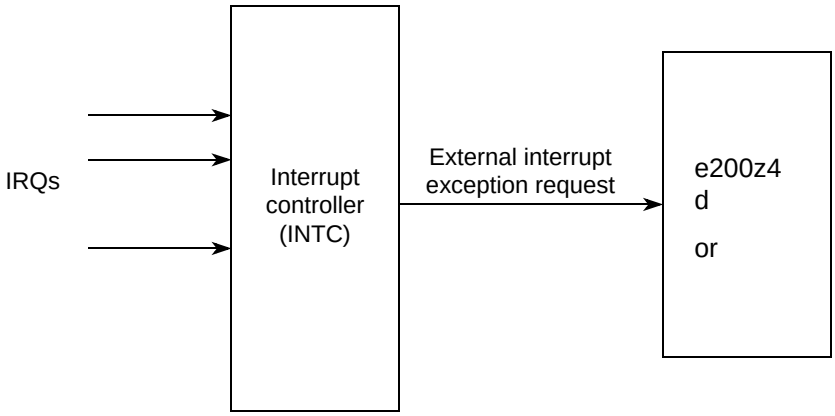
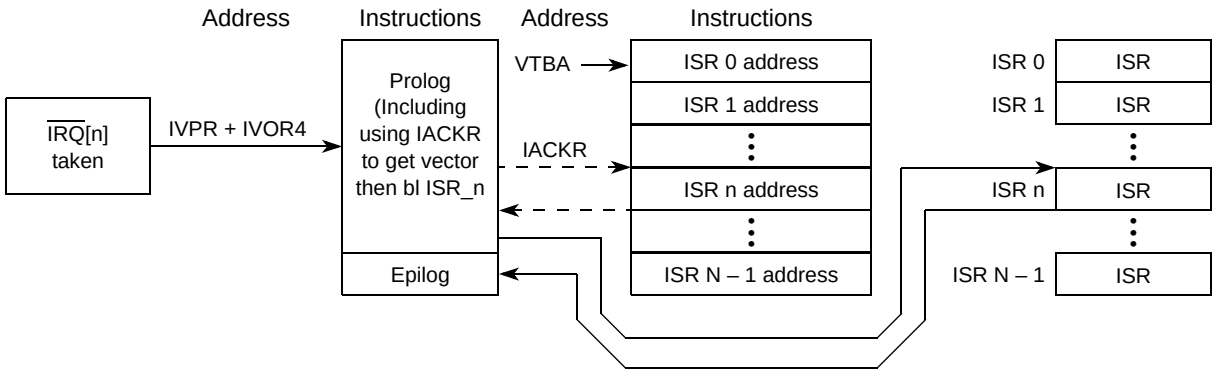


Figure 168. INTC Software Vector mode

Typical program flow for software vector mode is shown in *Figure 169*.



N is the maximum number of usable interrupt vectors, which equals 279, and includes 33 reserved IRQ vectors and eight software-settable IRQ vectors.

Figure 169. Program Flow–Software Vector mode

The common interrupt exception handler address is calculated by hardware as shown in *Figure 170* for the e200z0h core and *Figure 171* for the e200z4d core. The upper half of the interrupt vector prefix register (IVPR) is added to the offset contained in the external input

interrupt vector offset register (IVOR4). Note that since bits IVOR4[28:31] are not part of the offset value for the e200z4d, the vector offset must be located on a quad-word (16-byte) aligned location in memory. For the e200z0h core, the value of IVOR4 is hard coded to 0x040.

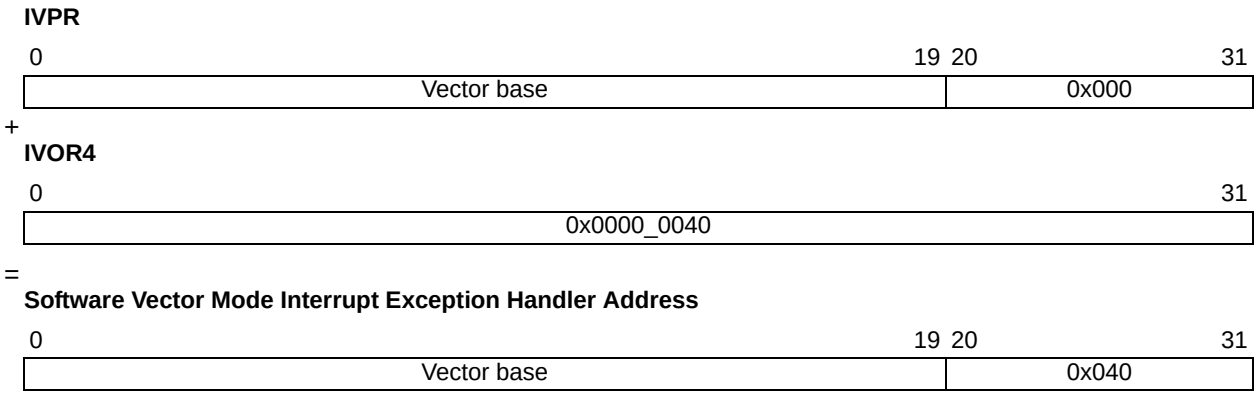


Figure 170. e200z0h Software Vector Mode: Interrupt Exception Handler Address Calculation

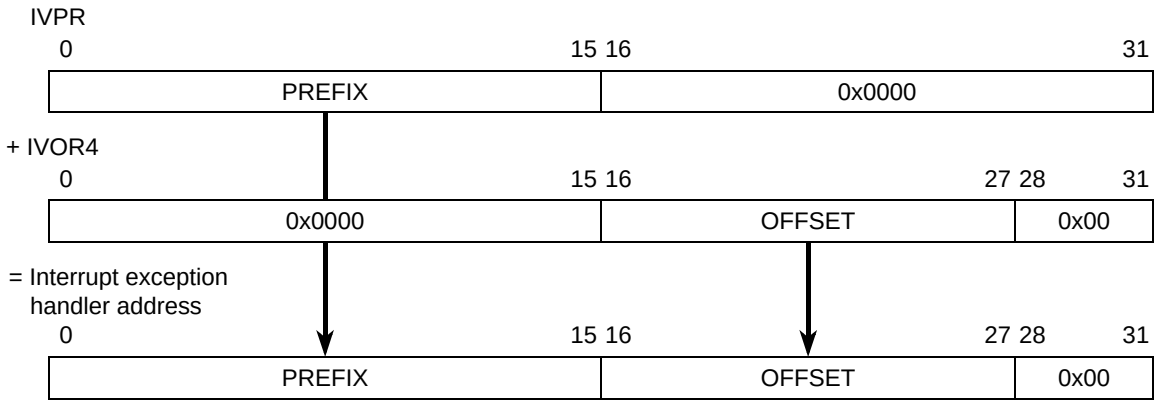


Figure 171. e200z4d Software Vector Mode: Interrupt Exception Handler Address Calculation

As shown in *Figure 169*, the common interrupt exception handler reads the INTC_IACKR_PRC*n* to determine the vector of the interrupt request source. The INTC_IACKR_PRC*n* register contains a 32-bit address for a vector table base address (VTBA) plus an offset to access the interrupt vector (INTVEC). The address is then used to branch to the corresponding routine for that peripheral or software interrupt source.

Reading the INTC_IACKR_PRC*n* acknowledges the INTC's interrupt request and negates the interrupt request to the processor. The interrupt request to the processor does not clear if a higher priority interrupt request arrives. Even in this case, INTVEC does not update to the higher priority request until the lower priority interrupt request is acknowledged by reading the INTC_IACKR_PRC*n*. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR_PRC*n*) onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the priority of the interrupt request. The INTC_CPR_PRC*n* masks any peripheral or software

settable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR_PRC*n* from generating an interrupt request to the processor.

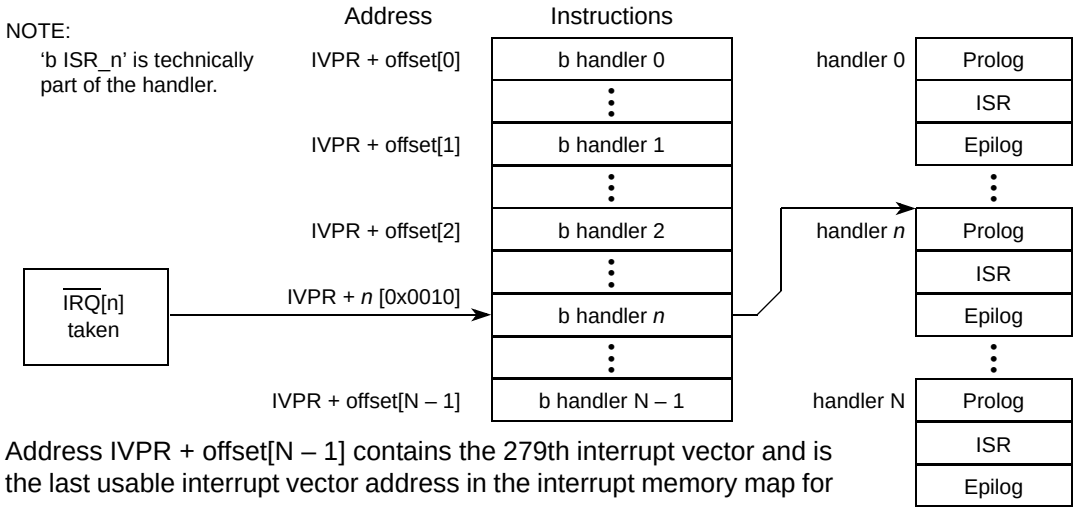
The interrupt exception handler must write to the end-of-interrupt register (INTC_EOIR_PRC*n*) to complete the operation. Writing to the INTC_EOIR_PRC*n* ends the servicing of the interrupt request. The INTC's LIFO is popped into the INTC_CPR_PRC*n*'s PRI field by writing to the INTC_EOIR_PRC*n*, and the size of a write does not affect the operation of the write. Those values and sizes written to this register neither update the INTC_EOIR_PRC*n* contents nor affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC*n*. The timing relationship between popping the LIFO and disabling recognition of external input has no restriction. The writes can happen in either order.

However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum stack depth at the cost of postponing the servicing of the next interrupt request.

19.4.2 Hardware Vector mode

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC can be optimized to support this goal through the hardware vector mode, where a unique vector is provided 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 has different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

Typical program flow for hardware vector mode is shown in *Figure 172*.



Address IVPR + offset[N – 1] contains the 279th interrupt vector and is the last usable interrupt vector address in the interrupt memory map for

N is the maximum number of usable interrupt vectors, which equals 279, and includes 33 reserved IRQ vectors and eight software-settable IRQ vectors.

Figure 172. Program Flow–Hardware Vector mode

In hardware vector mode, the interrupt exception handler address is specific to the peripheral or software settable interrupt source rather than being common to all of them. No IVOR is used. The interrupt exception handler address is calculated by hardware as shown in *Figure 173* for the e200z0h core and in *Figure 174* for the e200z4d core. The upper half of

the interrupt vector prefix register (IVPR) is added to an offset which corresponds to the peripheral or software interrupt source which caused the interrupt request. The offset matches the value in the Interrupt Vector field, INTC_IACKR_PRC*n*[INTVEC]. Each interrupt exception handler address is aligned on a quad word (16-byte) boundary for the e200z4d and on a word boundary (4-byte) for the e200z0h. IVOR4 is not used in this mode, and software does not need to read INTC_IACKR_PRC*n* to get the interrupt vector number.

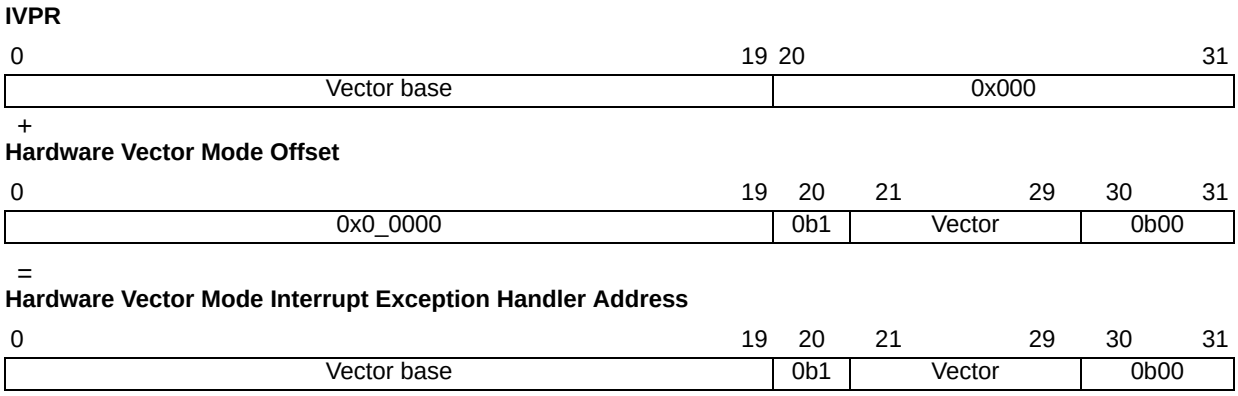


Figure 173. e200z0h Hardware Vector Mode: Interrupt Exception Handler Address Calculation

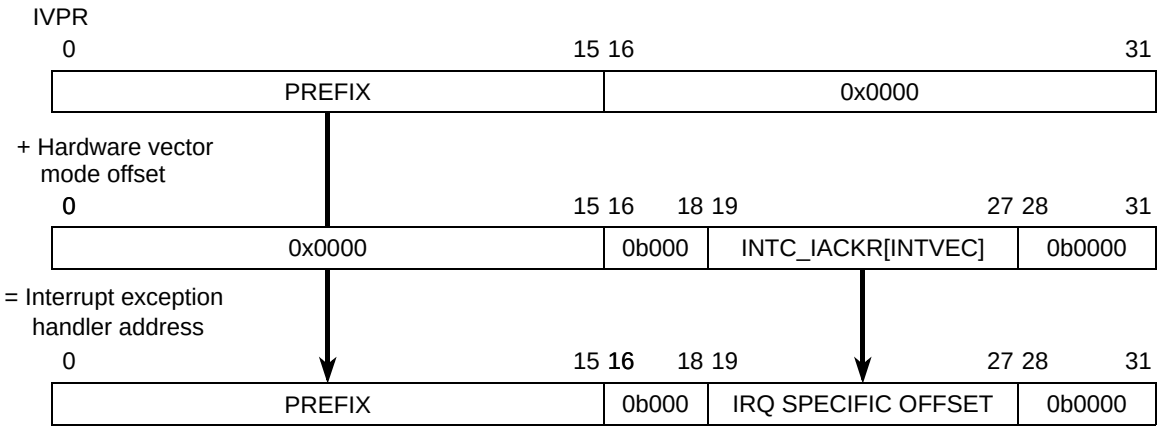


Figure 174. e200z4d Hardware Vector Mode: Interrupt Exception Handler Address Calculation

The processor negates INTC's interrupt request when automatically acknowledging the interrupt request. However, the interrupt request to the processor do not negate if a higher priority interrupt request arrives. Even in this case, the interrupt vector number does not update to the higher priority request until the lower priority request is acknowledged by the processor.

The assertion of the interrupt acknowledge signal pushes the PRI value in the INTC_CPR_PRC*n* onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the new priority.

19.5 Memory map and register description

19.5.1 Memory map

Table 171 shows the INTC memory map.

Table 171. INTC memory map

Base address: 0xFFF4_8000		
Address offset	**Register**	**Location**
0x0000	INTC Module Configuration Register (INTC_MCR)	*on page 434*
0x0004	Reserved	
0x0008	INTC Current Priority Register for Processor 0 (e200z4d) (INTC_CPR_PRC0)	*on page 435*
0x000C	INTC Current Priority Register for Processor 1 (e200z0h) (INTC_CPR_PRC1)	*on page 436*
0x0010	INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0)	*on page 436*
0x0014	INTC Interrupt Acknowledge Register for Processor 1 (e200z0h) (INTC_IACKR_PRC1)	*on page 438*
0x0018	INTC End of Interrupt Register for Processor 0 (e200z4d) (INTC_EOIR_PRC0)	*on page 439*
0x001C	INTC End of Interrupt Register for processor 1 (e200z0h) (INTC_EOIR_PRC1)	*on page 439*
0x0020–0x0027	INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)	*on page 439*
0x0028–0x003C	Reserved	
0x0040–0x0154	INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR276-278)(1)	*on page 440*

1. The PRI fields are “reserved” for peripheral interrupt requests whose vectors are labeled ‘Reserved’ in.

19.5.2 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_PRC0 and INTC_IACR_PRC1 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_PRC0 or INTC_EOIR_PRC1 does not affect the operation of the write.

19.5.2.1 INTC Module Configuration Register (INTC_MCR)

The module configuration register is used to configure options of the INTC.

Offset: 0x0000 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	VTES_	0	0	0	0	HVEN	0	0	VTES_	0	0	0	0	HVEN
W			PRC1					_PRC 1			PRC0					_PRC 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 175. INTC Module Configuration Register (INTC_MCR)

Table 172. INTC_MCR Field Descriptions

Field	Description
VTES_PRC1	For software mode only, the Vector Table Entry Size for Processor 1 (e200z0h). The VTES_PRC1 bit controls the number of 0s to the right of INTVEC_PRC1 in INTC_IACKR_PRC1. If the contents of INTC_IACKR_PRC1 are used as an address of an entry in a vector table, then the number of rightmost 0s will determine the size of each vector table entry. 0 4 bytes. 1 8 bytes.
HVEN_PRC1	Hardware Vector Enable for Processor 1 (e200z0h). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 19.4: Modes of operation*, for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.
VTES_PRC0	For software mode only, the Vector Table Entry Size for Processor 0 (e200z4d). The VTES_PRC0 bit controls the number of 0s to the right of INTVEC_PRC0 in INTC_IACKR_PRC0. If the contents of INTC_IACKR_PRC0 are used as an address of an entry in a vector table, then the number of rightmost 0s will determine the size of each vector table entry. 0 4 bytes. 1 8 bytes.
HVEN_PRC0	Hardware Vector Enable for Processor 0 (e200z4d). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 19.4: Modes of operation*, for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.

19.5.2.2 INTC Current Priority Register for Processor 0 (e200z4d) (INTC_CPR_PRC0)

Offset: 0x0008 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	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

Figure 176. INTC Current Priority Register for Processor 0 (e200z4d) (INTC_CPR_PRC0)

Table 173. INTC_CPR_PRC0 Field Descriptions

Field	Description
PRI	Priority PRI is the priority of the currently executing ISR according to the field values defined in *Table 174*.

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 19.10.4: 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 19.10.4.2: Ensuring Coherency for example code to ensure coherency.

Table 174. 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

Table 174. PRI values (continued)

PRI	Meaning
0100	Priority 4
0011	Priority 3
0010	Priority 2
0001	Priority 1
0000	Priority 0—lowest priority

19.5.2.3 INTC Current Priority Register for Processor 1 (e200z0h) (INTC_CPR_PRC1)

Offset: 0x000C 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	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

Figure 177. INTC Current Priority Register for Processor 1 (e200z0h) (INTC_CPR_PRC1)

The functionality of this register is the same as described for Processor 0 in *Section 19.5.2.2: INTC Current Priority Register for Processor 0 (e200z4d) (INTC_CPR_PRC0)*.

19.5.2.4 INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0)

Offset: 0x0010 Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VTBA_PRC0[20:5]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA_PRC0[4:0]				INTVEC_PRC0										0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 178. INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0) when INTC_MCR[VTES] = 0

Offset: 0x0010 Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VTBA_PRC00[19:4]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA_PRC0[3:0]				INTVEC_PRC0									0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 179. INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0) when INTC_MCR[VTES] = 1

Note: When the HVEN bit in the INTC module configuration register (INTC_MCR) is asserted, a read of the INTC_IACKR_PCRn has no side effects.

Table 175. INTC_IACKR_PRC0 Field Descriptions

Field	Description
VTBA_PRC0	Vector Table Base Address for Processor 0 (e200z4d) Can be the base address of a vector table of addresses of ISRs. The VTBA only uses the leftmost 20 bits when the VTES _PRC0 bit in INTC_MCR is asserted.
INTVEC_PRC0	Interrupt Vector for Processor 0 (e200z4d) 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_PRC0 is updated, whether the INTC_PRC0 is in software or hardware vector mode. **Note:** If INTC_MCR[VTES] = 1, then the INTVEC_PRC0 field is shifted left one position to bits 20–28. VTBA_PRC0 is then shortened by one bit to bits 0–19.

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_PRCx 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_PRCx does not have side effects in hardware vector mode.

Note: The INTC_IACKR_PRCn must not be read speculatively while in software vector mode. Therefore, for future compatibility, the TLB entry covering the INTC_IACKR_PRCn must be configured to be guarded.

Note: In software vector mode, the INTC_IACKR_PRCn must be read before setting MSR[EE]. No synchronization instruction is needed after reading the INTC_IACKR_PRCn and before setting MSR[EE].

Note: However, the time for the processor to recognize the assertion or negation of the external input to it is not defined by the book E architecture and can be greater than 0. Therefore, insert instructions between the reading of the INTC_IACKR_PRCn and the setting of MSR[EE] that consumes at least two processor clock cycles. This length of time allows the interrupt request negation to propagate through the processor before MSR[EE] is set.

19.5.2.5 INTC Interrupt Acknowledge Register for Processor 1 (e200z0h) (INTC_IACKR_PRC1)

Offset: 0x0014 Access: read/write

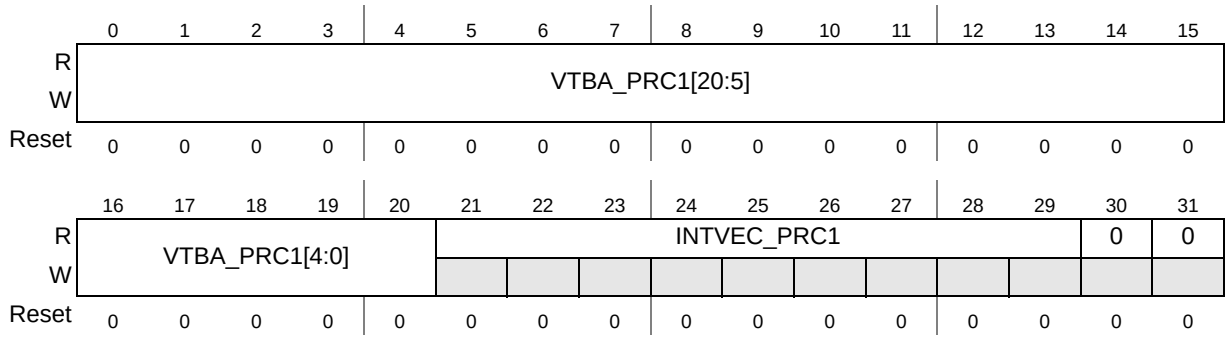


Figure 180. INTC Interrupt Acknowledge Register for Processor 1 (e200z0h) (INTC_IACKR_PRC1) when INTC_MCR[VTES] = 0

Offset: 0x0014 Access: read/write

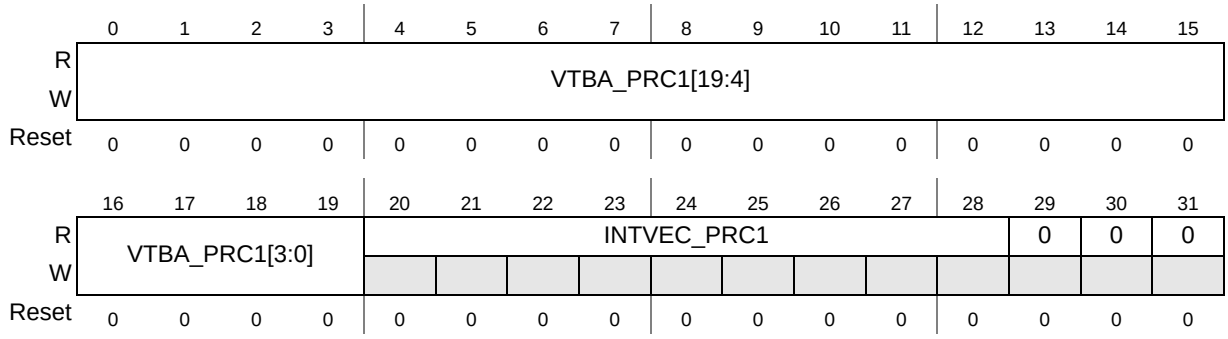


Figure 181. INTC Interrupt Acknowledge Register for Processor 1 (e200z0h) (INTC_IACKR_PRC1) when INTC_MCR[VTES] = 1

Table 176. INTC_IACKR_PRC1 Field Descriptions

Field	Description
VTBA_PRC1	The register's function is the same as described for processor 0 (e200z4d) in *Section 19.5.2.4: INTC Interrupt Acknowledge Register for Processor 0 (e200z4d) (INTC_IACKR_PRC0)*.
INTVEC_PRC1	

Note: When the HVEN bit in the INTC module configuration register (INTC_MCR) is asserted, a read of the INTC_IACKR_PCRn has no side effects.

19.5.2.6 INTC End of Interrupt Register for Processor 0 (e200z4d) (INTC_EOIR_PRC0)

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																																
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

Figure 182. INTC End of Interrupt Register for Processor 0 (e200z4d) (INTC_EOIR_PRC0)

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR_PRC0 is written, the priority last pushed on the LIFO is popped into INTC_CPR_PRC0. An exception to this behavior is described in *Section 19.4.2: Hardware Vector mode*. The values and size of data written to the INTC_EOIR_PRC0 are ignored. The values and sizes written to this register neither update the INTC_EOIR_PRC0 contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC0.

Reading the INTC_EOIR_PRC0 has no effect on the LIFO.

19.5.2.7 INTC End of Interrupt Register for processor 1 (e200z0h) (INTC_EOIR_PRC1)

Offset: 0x001C 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																																
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

Figure 183. INTC End of Interrupt Register for processor 1 (e200z0h) (INTC_EOIR_PRC1)

The register's function is the same as for processor 0 (e200z4d) as described in *Section 19.5.2.6: INTC End of Interrupt Register for Processor 0 (e200z4d) (INTC_EOIR_PRC0)*.

19.5.2.8 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

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	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 184. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

Offset: 0x0024 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	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

Figure 185. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

Table 177. 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.

19.5.2.9 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR276-278)

Offset: 0x0040 Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_SEL0		0	0	PRI0				PRC_SEL0 1		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	PRC_SEL2		0	0	PRI2				PRC_SEL3		0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 186. INTC Priority Select Register 0–3 (INTC_PSR[0:3])

Offset: 0x0154 Access: read/write

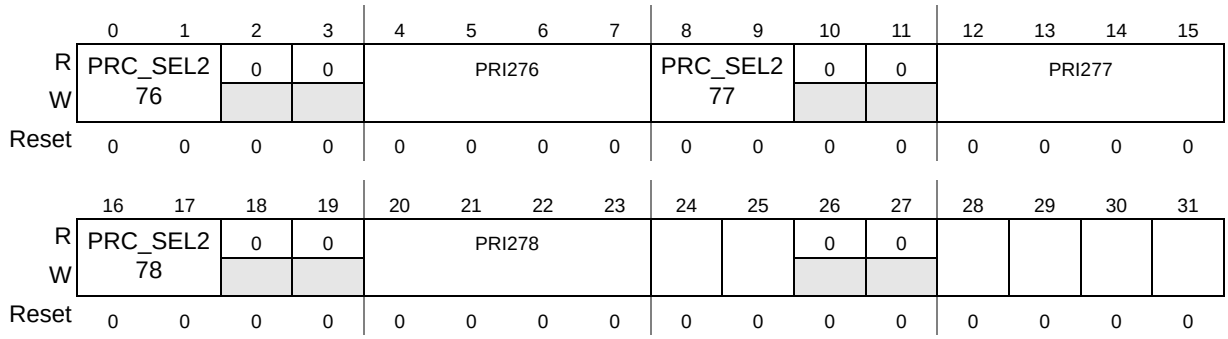


Figure 187. INTC Priority Select Register 276-278 (INTC_PSR[276:278])

Table 178. INTC_PSR0_3–INTC_PSR275-278 Field Descriptions

Field	Description
PRI	Priority Select PRI*x* selects the priority for interrupt requests. See *Section 19.6: Functional description*.
PRC_SEL	Processor Select. If an interrupt source is enabled, PRC_SEL*n* selects whether the interrupt request is to be sent to processor 0 (e200z4d), processor 1 (e200z0h), or both. See *Table 180*.

Table 179. INTC Priority Select Register Address Offsets

INTC_PSR*x*_*x*	Offset Address
INTC_PSR0_3	0x0040
INTC_PSR4_7	0x0044
INTC_PSR8_11	0x0048
INTC_PSR12_15	0x004C
INTC_PSR16_19	0x0050
INTC_PSR20_23	0x0054
INTC_PSR24_27	0x0058
INTC_PSR28_31	0x005C
INTC_PSR32_35	0x0060
INTC_PSR36_39	0x0064
INTC_PSR40_43	0x0068
INTC_PSR44_47	0x006C
INTC_PSR48_51	0x0070
INTC_PSR52_55	0x0074
INTC_PSR56_59	0x0078
INTC_PSR60_63	0x007C
INTC_PSR64_67	0x0080

INTC_PSR*x*_*x*	Offset Address
INTC_PSR140_143	0x00CC
INTC_PSR144_147	0x00D0
INTC_PSR148_151	0x00D4
INTC_PSR152_155	0x00D8
INTC_PSR156_159	0x00DC
INTC_PSR160_163	0x00E0
INTC_PSR164_167	0x00E4
INTC_PSR168_171	0x00E8
INTC_PSR172_175	0x00EC
INTC_PSR176_179	0x00F0
INTC_PSR180_183	0x00F4
INTC_PSR184_187	0x00F8
INTC_PSR188_191	0x00FC
INTC_PSR192_195	0x0100
INTC_PSR196_199	0x0104
INTC_PSR200_203	0x0108
INTC_PSR204_207	0x010C

Table 179. INTC Priority Select Register Address Offsets (continued)

INTC_PSR*x*_*x*	Offset Address
INTC_PSR68_71	0x0084
INTC_PSR72_75	0x0088
INTC_PSR76_79	0x008C
INTC_PSR80_83	0x0090
INTC_PSR84_87	0x0094
INTC_PSR88_91	0x0098
INTC_PSR92_95	0x009C
INTC_PSR96_99	0x00A0
INTC_PSR100_103	0x00A4
INTC_PSR104_107	0x00A8
INTC_PSR108_111	0x00AC
INTC_PSR112_115	0x00B0
INTC_PSR116_119	0x00B4
INTC_PSR120_123	0x00B8
INTC_PSR124_127	0x00BC
INTC_PSR128_131	0x00C0
INTC_PSR132_135	0x00C4
INTC_PSR136_139	0x00C8

INTC_PSR*x*_*x*	Offset Address
INTC_PSR208_211	0x0110
INTC_PSR212_215	0x0114
INTC_PSR216_219	0x0118
INTC_PSR220_223	0x011C
INTC_PSR224_227	0x0120
INTC_PSR228_231	0x0124
INTC_PSR232_235	0x0128
INTC_PSR236_239	0x012C
INTC_PSR240_243	0x0130
INTC_PSR244_247	0x0134
INTC_PSR248_251	0x0138
INTC_PSR252_255	0x013C
INTC_PSR256_259	0x0140
INTC_PSR260_263	0x0144
INTC_PSR264_267	0x0148
INTC_PSR268_271	0x014C
INTC_PSR272_275	0x0150
INTC_PSR276_278	0x0154

The priority select registers support the selection of an individual priority for each source of interrupt request, and whether the interrupt request is to be sent to processor 0 (e200z4d), processor 1, (e200z0h) or both. The unique vector of each peripheral or software settable interrupt request determines which INTC_PSR*x*_*x* is assigned to that interrupt request. The software settable interrupt requests 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0_3 and INTC_PSR4_7, respectively. The peripheral interrupt requests are assigned vectors 8–278, and their priorities are configured in INTC_PSR8_11 through INTC_PSR275_278, respectively (see *Table 179*).

Note: The PRC_SELx or PRIx field of an INTC_PSRx_x must not be modified while the corresponding peripheral or software settable interrupt request is asserted.

Table 180. Selected Processor for Interrupt Request

PRC_SEL*x*[0:1]	Meaning
00	Interrupt request sent to processor 0 (e200z4d)
01	Interrupt request sent to both processors
10	Reserved
11	Interrupt request sent to processor 1 (e200z0h)

19.6 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose PRI*n* value in INTC_PSR0–INTC_PSR211 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 PRI*n* value in INTC_PSR*n*. 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.The INTC has two types of interrupt requests, peripheral and software settable. The assignments between the interrupt requests from the modules to the vectors for input to the CPU are shown in *Table 181*, *Table 182*, *Table 183*.

Table 181. e200z0h core interrupts

IRQ#	Offset	Size [Byte]	Resource	Module
Section A (e200z0h Core Section)				
—	0x0000	16	Critical Input – NMI[0] (WKPU)	Core WKPU
—	0x0010	16	Machine check / NMI – NMI[0] (WKPU)	Core WKPU
—	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	Reserved	Core

Table 182. e200z4d core interrupts

IRQ #	Offset	Size [Byte]	Resource	Module
Section A (e200z4d Core Section)				
—	IVOR0	16	Critical Input (INTC software vector mode) – NMI[1] (WKPU)	Core WKPU
—	IVOR1	16	Machine check / NMI – NMI[1] (WKPU)	Core WKPU
—	IVOR2	16	Data Storage	Core
—	IVOR3	16	Instruction Storage	Core
—	IVOR4	16	External Input (INTC software vector mode)	Core
—	IVOR5	16	Alignment	Core
—	IVOR6	16	Program	Core
—	IVOR7	16	Floating-Point unavailable	Core
—	IVOR8	16	System call	Core
—	IVOR9	16	Unused	Core
—	IVOR10	16	Decrementer	Core
—	IVOR11	16	Fixed Interval Timer	Core
—	IVOR12	16	Watchdog Timer	Core
—	IVOR13	16	Data TLB Error	Core
—	IVOR14	16	Instruction TLB Error	Core
—	IVOR15	16	Debug	Core
—	IVOR16-31	256	Unused	Core
—	IVOR32	16	SPE Unavailable Exception	Core
—	IVOR33	16	EFP Data Exception	Core
—	IVOR34	16	EFP Round Exception	Core

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
Section B (On-Platform Peripherals)				
0	0x0800	0x0000	Software settable flag 0	Software
1	0x0804	0x0010	Software settable flag 1	
2	0x0808	0x0020	Software settable flag 2	
3	0x080C	0x0030	Software settable flag 3	
4	0x0810	0x0040	Software settable flag 4	
5	0x0814	0x0050	Software settable flag 5	
6	0x0818	0x0060	Software settable flag 6	
7	0x081C	0x0070	Software settable flag 7	
8	0x0820	0x0080	Reserved	
9	0x0824	0x0090	Platform Flash Bank 0 Abort | Platform Flash Bank 0 Stall | Platform Flash Bank 1 Abort | Platform Flash Bank 1 Stall | Platform Flash Bank 2 Abort | Platform Flash Bank 2 Stall |	ECSM

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
10	0x0828	0x00A0	Combined Error [15:0]	eDMA
11	0x082C	0x00B0	Channel 0	
12	0x0830	0x00C0	Channel 1	
13	0x0834	0x00D0	Channel 2	
14	0x0838	0x00E0	Channel 3	
15	0x083C	0x00F0	Channel 4	
16	0x0840	0x0100	Channel 5	
17	0x0844	0x0110	Channel 6	
18	0x0848	0x0120	Channel 7	
19	0x084C	0x0130	Channel 8	
20	0x0850	0x0140	Channel 9	
21	0x0854	0x0150	Channel 10	
22	0x0858	0x0160	Channel 11	
23	0x085C	0x0170	Channel 12	
24	0x0860	0x0180	Channel 13	
25	0x0864	0x0190	Channel 14	
26	0x0868	0x01A0	Channel 15	
27	0x086C	0x01B0	Reserved	
28	0x0870	0x01C0	Timeout	Software Watchdog (SWT)
29	0x0874	0x01D0	Reserved	
30	0x0878	0x01E0	Match on channel 0	STM
31	0x087C	0x01F0	Match on channel 1	
32	0x0880	0x0200	Match on channel 2	
33	0x0884	0x0210	Match on channel 3	
34	0x0888	0x0220	Reserved	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
35	0x088C	0x0230	ECC_DBD_PlatformFlash | ECC_DBD_PlatformRAM	ECSM - Platform ECC Double Bit Detection
36	0x0890	0x0240	ECC_SBC_PlatformFlash | ECC_SBC_PlatformRAM	ECSM - Platform ECC Single Bit Correction
37	0x0894	0x0250	Reserved	
Section C				
38	0x0898	0x0260	RTC	Real Time Counter (RTC/API)
39	0x089C	0x0270	API	
40	0x08A0	0x0280	Reserved	Reserved
41	0x08A4	0x0290	SIU External IRQ_0	System Integration Unit Lite (SIUL)
42	0x08A8	0x02A0	SIU External IRQ_1	
43	0x08AC	0x02B0	SIU External IRQ_2	
44	0x08B0	0x02C0	Reserved	
45	0x08B4	0x02D0	Reserved	
46	0x08B8	0x02E0	WakeUp_IRQ_0	WakeUp Unit (WKPU)
47	0x08BC	0x02F0	WakeUp_IRQ_1	
48	0x08C0	0x0300	WakeUp_IRQ_2	
49	0x08C4	0x0310	WakeUp_IRQ_3	
50	0x08C8	0x0320	Reserved	
51	0x08CC	0x0330	Safe Mode Interrupt	MC_ME
52	0x08D0	0x0340	Mode Transition Interrupt	
53	0x08D4	0x0350	Invalid Mode Interrupt	
54	0x08D8	0x0360	Invalid Mode Config	
55	0x08DC	0x0370	Reserved	
56	0x08E0	0x0380	Functional and destructive reset alternate event interrupt (ipi_int)	MC_RGM

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
57	0x08E4	0x0390	XOSC counter expired (ipi_int_osc)	FXOSC
58	0x08E8	0x03A0	PIT_RTI	Periodic Interrupt Timer- Real Time Interrupt
59	0x08EC	0x03B0	PITimer Channel 0	PIT_RTI
60	0x08F0	0x03C0	PITimer Channel 1	
61	0x08F4	0x03D0	PITimer Channel 2	
62	0x08F8	0x03E0	ADC_EOC	Analog to Digital Converter 0 (ADC0)
63	0x08FC	0x03F0	Reserved	
64	0x0900	0x0400	ADC_WD	Analog to Digital Converter 0 (ADC0)
65	0x0904	0x0410	FLEXCAN_ESR[ERR_INT]	FlexCAN 0 (CAN0)
66	0x0908	0x0420	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	
67	0x090C	0x0430	Reserved	
68	0x0910	0x0440	FLEXCAN_BUF_00_03	FlexCAN 0 (CAN0)
69	0x0914	0x0450	FLEXCAN_BUF_04_07	
70	0x0918	0x0460	FLEXCAN_BUF_08_11	
71	0x091C	0x0470	FLEXCAN_BUF_12_15	
72	0x0920	0x0480	FLEXCAN_BUF_16_31	
73	0x0924	0x0490	FLEXCAN_BUF_32_63	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
74	0x0928	0x04A0	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 0
75	0x092C	0x04B0	DSPI_SR[EOQF]	
76	0x0930	0x04C0	DSPI_SR[TFFF]	
77	0x0934	0x04D0	DSPI_SR[TCF] DSPI_SR[DDIF]	
78	0x0938	0x04E0	DSPI_SR[RFDF]	
79	0x093C	0x04F0	LINFlexD_RXI	LINFlexD_0
80	0x0940	0x0500	LINFlexD_TXI	
81	0x0944	0x0510	LINFlexD_ERR	
82	0x0948	0x0520	ADC_EOC	Analog to Digital Converter 1 (ADC1)
83	0x094C	0x0530	Reserved	
84	0x0950	0x0540	ADC_WD	Analog to Digital Converter 1 (ADC1)
85	0x0954	0x0550	FLEXCAN_ESR[ERR_INT]	FlexCAN 1 (CAN1)
86	0x0958	0x0560	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	
87	0x095C	0x0570	Reserved	
88	0x0960	0x0580	FLEXCAN_BUF_00_03	FlexCAN 1 (CAN1)
89	0x0964	0x0590	FLEXCAN_BUF_04_07	
90	0x0968	0x05A0	FLEXCAN_BUF_08_11	
91	0x096C	0x05B0	FLEXCAN_BUF_12_15	
92	0x0970	0x05C0	FLEXCAN_BUF_16_31	
93	0x0974	0x05D0	FLEXCAN_BUF_32_63	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
94	0x0978	0x05E0	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 1
95	0x097C	0x05F0	DSPI_SR[EOQF]	
96	0x0980	0x0600	DSPI_SR[TFFF]	
97	0x0984	0x0610	DSPI_SR[TCF] DSPI_SR[DDIF]	
98	0x0988	0x0620	DSPI_SR[RFDF]	
99	0x098C	0x0630	LINFlexD_RXI	LINFlexD_1
100	0x0990	0x0640	LINFlexD_TXI	
101	0x0994	0x0650	LINFlexD_ERR	
102	0x0998	0x0660	Reserved	
103	0x099C	0x0670	Reserved	
104	0x09A0	0x0680	Reserved	
105	0x09A4	0x0690	FLEXCAN_ESR[ERR_INT]	FlexCAN 2 (CAN2)
106	0x09A8	0x06A0	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	
107	0x09AC	0x06B0	Reserved	
108	0x09B0	0x06C0	FLEXCAN_BUF_00_03	FlexCAN 2 (CAN2)
109	0x09B4	0x06D0	FLEXCAN_BUF_04_07	
110	0x09B8	0x06E0	FLEXCAN_BUF_08_11	
111	0x09BC	0x06F0	FLEXCAN_BUF_12_15	
112	0x09C0	0x0700	FLEXCAN_BUF_16_31	
113	0x09C4	0x0710	FLEXCAN_BUF_32_63	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
114	0x09C8	0x0720	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 2
115	0x09CC	0x0730	DSPI_SR[EOQF]	
116	0x09D0	0x0740	DSPI_SR[TFFF]	
117	0x09D4	0x0750	DSPI_SR[TCF] DSPI_SR[DDIF]	
118	0x09D8	0x0760	DSPI_SR[RFDF]	
119	0x09DC	0x0770	LINFlexD_RXI	LINFlexD_2
120	0x09E0	0x0780	LINFlexD_TXI	
121	0x09E4	0x0790	LINFlexD_ERR	
122	0x09E8	0x07A0	LINFlexD_RXI	LINFlexD_3
123	0x09EC	0x07B0	LINFlexD_TXI	
124	0x09F0	0x07C0	LINFlexD_ERR	
125	0x09F4	0x07D0	IIC_SR[IBAL] IIC_SR[TCF] IIC_SR[IAAS]	Inter-IC Bus Interface Controller (I2C)
126	0x09F8	0x07E0	Reserved	
127	0x09FC	0x07F0	PITimer Channel 3	PIT_RTI
128	0x0A00	0x0800	PITimer Channel 4	
129	0x0A04	0x0810	PITimer Channel 5	
130	0x0A08	0x0820	PITimer Channel 6	
131	0x0A0C	0x0830	PITimer Channel 7	
132	0x0A10	0x0840	Reserved	
133	0x0A14	0x0850	Reserved	
134	0x0A18	0x0860	Reserved	
135	0x0A1C	0x0870	Reserved	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
136	0x0A20	0x0880	Reserved	
137	0x0A24	0x0890	Reserved	
138	0x0A28	0x08A0	Reserved	
139	0x0A2C	0x08B0	Reserved	
140	0x0A30	0x08C0	Reserved	
141	0x0A34	0x08D0	EMIOS_GFR[F0,F1]	Enhanced Modular I/O Subsystem 0 (eMIOS0)
142	0x0A38	0x08E0	EMIOS_GFR[F2,F3]	
143	0x0A3C	0x08F0	EMIOS_GFR[F4,F5]	
144	0x0A40	0x0900	EMIOS_GFR[F6,F7]	
145	0x0A44	0x0910	EMIOS_GFR[F8,F9]	
146	0x0A48	0x0920	EMIOS_GFR[F10,F11]	
147	0x0A4C	0x0930	EMIOS_GFR[F12,F13]	
148	0x0A50	0x0940	EMIOS_GFR[F14,F15]	
149	0x0A54	0x0950	EMIOS_GFR[F16,F17]	
150	0x0A58	0x0960	EMIOS_GFR[F18,F19]	
151	0x0A5C	0x0970	EMIOS_GFR[F20,F21]	
152	0x0A60	0x0980	EMIOS_GFR[F22,F23]	
153	0x0A64	0x0990	EMIOS_GFR[F24,F25]	
154	0x0A68	0x09A0	EMIOS_GFR[F26,F27]	
155	0x0A6C	0x09B0	EMIOS_GFR[F28,F29]	
156	0x0A70	0x09C0	EMIOS_GFR[F30,F31]	
Section D (Device specific vectors)				

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
157	0x0A74	0x09D0	EMIOS_GFR[F0,F1]	Enhanced Modular I/O Subsystem 1 (eMIOS1)
158	0x0A78	0x09E0	EMIOS_GFR[F2,F3]	
159	0x0A7C	0x09F0	EMIOS_GFR[F4,F5]	
160	0x0A80	0x0A00	EMIOS_GFR[F6,F7]	
161	0x0A84	0x0A10	EMIOS_GFR[F8,F9]	
162	0x0A88	0x0A20	EMIOS_GFR[F10,F11]	
163	0x0A8C	0x0A30	EMIOS_GFR[F12,F13]	
164	0x0A90	0x0A40	EMIOS_GFR[F14,F15]	
165	0x0A94	0x0A50	EMIOS_GFR[F16,F17]	
166	0x0A98	0x0A60	EMIOS_GFR[F18,F19]	
167	0x0A9C	0x0A70	EMIOS_GFR[F20,F21]	
168	0x0AA0	0x0A80	EMIOS_GFR[F22,F23]	
169	0x0AA4	0x0A90	EMIOS_GFR[F24,F25]	
170	0x0AA8	0x0AA0	EMIOS_GFR[F26,F27]	
171	0x0AAC	0x0AB0	EMIOS_GFR[F28,F29]	
172	0x0AB0	0x0AC0	EMIOS_GFR[F30,F31]	
173	0x0AB4	0x0AD0	FLEXCAN_ESR	FLEXCAN 3
174	0x0AB8	0x0AE0	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	
175	0x0ABC	0x0AF0	Reserved	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
176	0x0AC0	0x0B00	FLEXCAN_BUF_0_3	FLEXCAN 3
177	0x0AC4	0x0B10	FLEXCAN_BUF_4_7	
178	0x0AC8	0x0B20	FLEXCAN_BUF_8_11	
179	0x0ACC	0x0B30	FLEXCAN_BUF_12_15	
180	0x0AD0	0x0B40	FLEXCAN_BUF_16_31	
181	0x0AD4	0x0B50	FLEXCAN_BUF_32_63	
182	0x0AD8	0x0B60	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 3
183	0x0ADC	0x0B70	DSPI_SR[EOQF]	
184	0x0AE0	0x0B80	DSPI_SR[TFFF]	
185	0x0AE4	0x0B90	DSPI_SR[TCF] DSPI_SR[DDIF]	
186	0x0AE8	0x0BA0	DSPI_SR[RFDF]	
187	0x0AEC	0x0BB0	LINFlexD_RXI	LINFlexD_4
188	0x0AF0	0x0BC0	LINFlexD_TXI	
189	0x0AF4	0x0BD0	LINFlexD_ERR	
190	0x0AF8	0x0BE0	FLEXCAN_ESR	FLEXCAN 4
191	0x0AFC	0x0BF0	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	
192	0x0B00	0x0C00	Reserved	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
193	0x0B04	0x0C10	FLEXCAN_BUF_0_3	FLEXCAN 4
194	0x0B08	0x0C20	FLEXCAN_BUF_4_7	
195	0x0B0C	0x0C30	FLEXCAN_BUF_8_11	
196	0x0B10	0x0C40	FLEXCAN_BUF_12_15	
197	0x0B14	0x0C50	FLEXCAN_BUF_16_31	
198	0x0B18	0x0C60	FLEXCAN_BUF_32_63	
199	0x0B1C	0x0C70	LINFlexD_RXI	LINFlexD_5
200	0x0B20	0x0C80	LINFlexD_TXI	
201	0x0B24	0x0C90	LINFlexD_ERR	
202	0x0B28	0x0CA0	FLEXCAN_ESR	FLEXCAN 5
203	0x0B2C	0x0CB0	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	
204	0x0B30	0x0CC0	Reserved	
205	0x0B34	0x0CD0	FLEXCAN_BUF_0_3	FLEXCAN 5
206	0x0B38	0x0CE0	FLEXCAN_BUF_4_7	
207	0x0B3C	0x0CF0	FLEXCAN_BUF_8_11	
208	0x0B40	0x0D00	FLEXCAN_BUF_12_15	
209	0x0B44	0x0D10	FLEXCAN_BUF_16_31	
210	0x0B48	0x0D20	FLEXCAN_BUF_32_63	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
211	0x0B4C	0x0D30	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 4
212	0x0B50	0x0D40	DSPI_SR[EOQF]	
213	0x0B54	0x0D50	DSPI_SR[TFFF]	
214	0x0B58	0x0D60	DSPI_SR[TCF] DSPI_SR[DDIF]	
215	0x0B5C	0x0D70	DSPI_SR[RFDF]	
216	0x0B60	0x0D80	LINFlexD_RXI	LINFlexD_6
217	0x0B64	0x0D90	LINFlexD_TXI	
218	0x0B68	0x0DA0	LINFlexD_ERR	
219	0x0B6C	0x0DB0	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 5
220	0x0B70	0x0DC0	DSPI_SR[EOQF]	
221	0x0B74	0x0DD0	DSPI_SR[TFFF]	
222	0x0B78	0x0DE0	DSPI_SR[TCF] DSPI_SR[DDIF]	
223	0x0B7C	0x0DF0	DSPI_SR[RFDF]	
224	0x0B80	0x0E00	LINFlexD_RXI	LINFlexD_7
225	0x0B84	0x0E10	LINFlexD_TXI	
226	0x0B88	0x0E20	LINFlexD_ERR	
227	0x0B8C	0x0E30	LINFlexD_RXI	LINFlexD_8
228	0x0B90	0x0E40	LINFlexD_TXI	
229	0x0B94	0x0E50	LINFlexD_ERR	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
230	0x0B98	0x0E60	LINFlexD_RXI	LINFlexD_9
231	0x0B9C	0x0E70	LINFlexD_TXI	
232	0x0BA0	0x0E80	LINFlexD_ERR	
233	0x0BA4	0x0E90	32KXOSC counter expired	32KFXOSC
234	0x0BA8	0x0EA0	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 6
235	0x0BAC	0x0EB0	DSPI_SR[EOQF]	
236	0x0BB0	0x0EC0	DSPI_SR[TFFF]	
237	0x0BB4	0x0ED0	DSPI_SR[TCF] DSPI_SR[DDIF]	
238	0x0BB8	0x0EE0	DSPI_SR[RFDF]	
239	0x0BBC	0x0EF0	DSPI_SR[TFUF] DSPI_SR[RFOF] DSPI_SR[DPEF] DSPI_SR[SPEF]	DSPI 7
240	0x0BC0	0x0F00	DSPI_SR[EOQF]	
241	0x0BC4	0x0F10	DSPI_SR[TFFF]	
242	0x0BC8	0x0F20	DSPI_SR[TCF] DSPI_SR[DDIF]	
243	0x0BCC	0x0F30	DSPI_SR[RFDF]	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
244	0x0BD0	0x0F40	EIR[TXF]	FEC
245	0x0BD4	0x0F50	EIR[RXF]	
246	0x0BD8	0x0F60	EIR[HBERR] EIR[BABR] EIR[BABT] EIR[GRA] EIR[TXB] EIR[RXB] EIR[MII] EIR[EBERR] EIR[LC] EIR[RL] EIR[UN]	
247	0x0BDC	0x0F70	CIFRR.FAFAIF	FlexRay Controller (FlexRay)
248	0x0BE0	0x0F80	CIFRR.FAFBIF	
249	0x0BE4	0x0F90	CIFRR.WUPIF	
250	0x0BE8	0x0FA0	CIFRR.PRIF	
251	0x0BEC	0x0FB0	CIFRR.CHIF	
252	0x0BF0	0x0FC0	CIFRR.TBIF	
253	0x0BF4	0x0FD0	CIFRR.RBIF	
254	0x0BF8	0x0FE0	CIFRR.MIF	
255	0x0BFC	0x0FF0	LRCEIF | DRCEIF	
256	0x0C00	0x1000	LRNEIF | DRNEIF	
257	0x0C04	0x1010	Semaphore Int 0	Semaphore
258	0x0C08	0x1020	Semaphore Int 1	
259	0x0C0C	0x1030	CSE Complete	CSE
260	0x0C10	0x1040	Reserved	

Table 183. SPC564Bxx and SPC56ECxx Interrupt Vector Table (continued)

IRQ #	e200z0h hardware vector mode offset (vector size is 4B)	e200z4d hardware vector mode offset (vector size is 16B)	Resource	Module
261	0x0C14	0x1050	Reserved	
262	0x0C18	0x1060	Combined Error [31:16]	DMA
263	0x0C19	0x1070	Channel 16	
264	0x0C1A	0x1080	Channel 17	
265	0x0C1B	0x1090	Channel 18	
266	0x0C1C	0x10A0	Channel 19	
267	0x0C1D	0x10B0	Channel 20	
268	0x0C1E	0x10C0	Channel 21	
269	0x0C1F	0x10D0	Channel 22	
270	0x0C20	0x10E0	Channel 23	
271	0x0C21	0x10F0	Channel 24	
272	0x0C22	0x1100	Channel 25	
273	0x0C23	0x1110	Channel 26	
274	0x0C24	0x1120	Channel 27	
275	0x0C25	0x1130	Channel 28	
276	0x0C26	0x1140	Channel 39	
277	0x0C27	0x1150	Channel 30	
278	0x0C28	0x1160	Channel 31	

19.7 SIUL external interrupts

SPC564Bxx and SPC56ECxx supports three interrupt vectors dedicated to the 24 external interrupts.

19.8 Wakeup line interrupts

SPC564Bxx and SPC56ECxx supports four interrupts vectors dedicated to the 32 wakeup lines into wake-up unit (30 external pins + 1 internal from RTC + 1 internal from API).

19.9 Non-maskable interrupt (NMI)

SPC564Bxx and SPC56ECxx supports two NMI inputs which are mapped on the NMI[0;1] input port of the Wake Up unit.

SPC564Bxx and SPC56ECxx NMI connection is as described in the *Figure 188*.

There are two NMI inputs mapped from I/Os to the wake up unit NMI inputs:

PA1 -> NMI[0] -> e200z0h CI (IVOR0) and MC_NMI (IVOR1).

PA2 -> NMI[1] -> e200z4d C I(IVOR0) and MC_NMI (IVOR1).

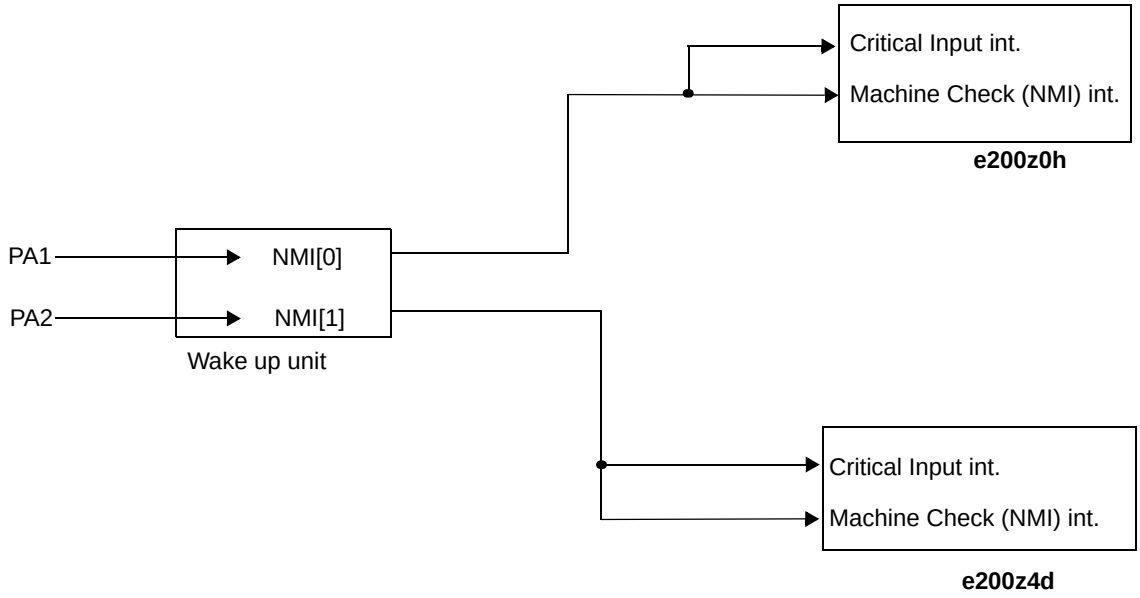


Figure 188. Cores NMI sources

Each NMI input of the wake up unit is as well a source of the Interrupt Vector 0.

19.9.1 Interrupt Request sources

The INTC has two types of interrupt requests, peripheral and software configurable. These interrupt requests can assert on any clock cycle.

19.9.1.1 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 SIUL.

19.9.1.2 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.

19.9.1.3 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 170*).

19.9.2 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_PSR276-278). 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.

19.9.2.1 Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in *Figure 167* 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_PRC0 or INTC_IACKR_PRC1), and if in hardware vector mode, for the interrupt vector provided to the processor.

19.9.2.1.1 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.

19.9.2.1.2 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.

19.9.2.1.3 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.

19.9.2.1.4 Priority Comparator Subblock

The priority comparator subblock compares the highest priority output from the priority arbitrator subblock with PRI in INTC_CPR_PRC0 or INTC_CPR_PRC1. 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_PRC0 or INTC_CPR_PRC1 or the PRI value in INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered below this highest priority. This highest priority then becomes the new priority which will be written to PRI in INTC_CPR_PRC0 or INTC_CPR_PRC1 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_PRC0 or INTC_CPR_PRC1.

Another function of the priority comparator subblock is to signal an update of the INTC_IACKR_PRC0 and INTC_IACKR_PRC1 with the vector number of the first interrupt that arrives that has a priority higher than the current priority. Once the vector number and priority are captured, they cannot be superseded by a higher priority interrupt until an update of the INTC_CPR_PRC0 or INTC_CPR_PRC1 occurs. An optional design definition can be used at module instantiation to change this behavior such that higher priority interrupts can supersede the previously captured interrupt vector number and priority up until a hardware or software interrupt acknowledge is processed.

19.9.2.2 Last-In First-Out (LIFO)

The LIFO stores the preempted PRI values from the INTC_CPR_PRC0 or INTC_CPR_PRC1. 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_PRC0 or INTC_CPR_PRC1 does not need to be loaded from the INTC_CPR_PRC0 or INTC_CPR_PRC1 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_PRC0 or INTC_CPR_PRC1.

The PRI value in the INTC_CPR_PRC0 or INTC_CPR_PRC1 is pushed onto the LIFO when the INTC_IACKR_PRC0 or INTC_IACKR_PRC1 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_PRC0 or INTC_CPR_PRC1 whenever the INTC_EOIR_PRC0 or INTC_EOIR_PRC1 is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR_PRC0 or INTC_CPR_PRC1 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.

19.9.3 Handshaking with processor

19.9.3.1 Software Vector Mode Handshaking

This section describes handshaking in software vector mode.

19.9.3.1.1 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 189*. 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_PRC0 or INTC_CPR_PRC1 register, it asserts the interrupt request to the processor. The INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 register 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 19.4.1: Software Vector mode*.

19.9.3.1.2 End of Interrupt Exception Handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR_PRC0 or INTC_EOIR_PRC1) must be written.When written, the associated LIFO is popped so the preempted priority is restored into PRI of the INTC_CPR_PRC0 or INTC_CPR_PRC1. 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_PRC0 or INTC_EOIR_PRC1.

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_PRC0 or INTC_CPR_PRC1 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.

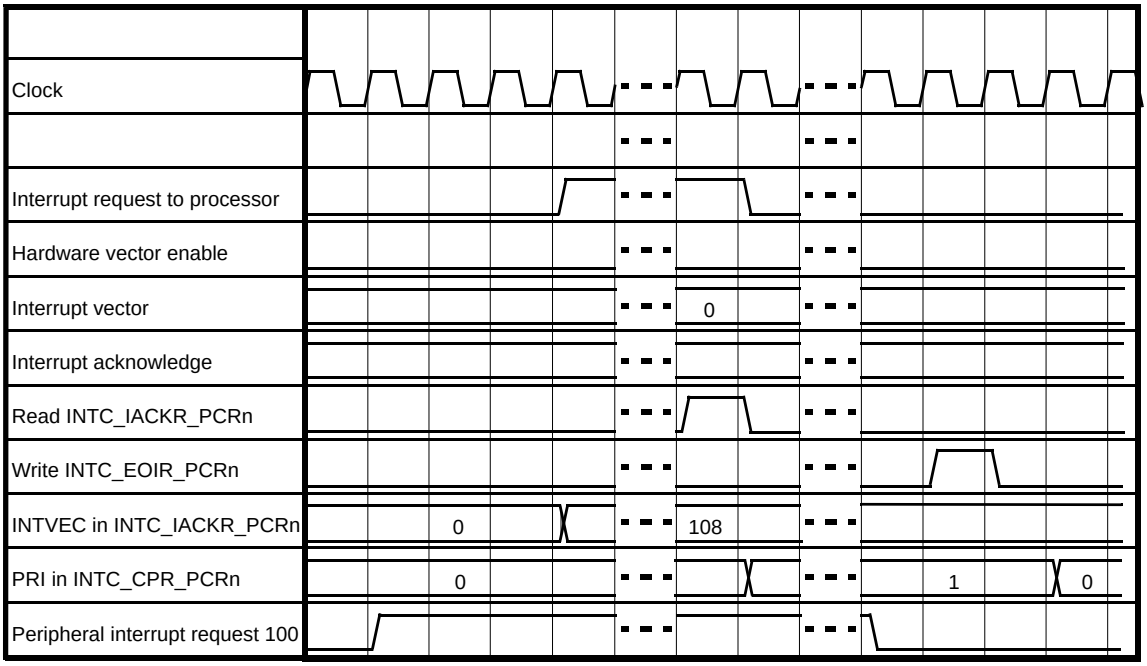


Figure 189. Software Vector Mode Handshaking Timing Diagram

19.9.3.2 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 190*. 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_PRC0 or INTC_CPR_PRC1, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR_PRC0 or INTC_IACKR_PRC1 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_PRC0 or INTC_IACKR_PRC1. The rest of the handshaking is described in *Section 19.10.2.2: Hardware vector mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR_PRC0 or INTC_EOIR_PRC1, is the same as in software vector mode. Refer to *Section 19.9.3.1.2: End of Interrupt Exception Handler*.

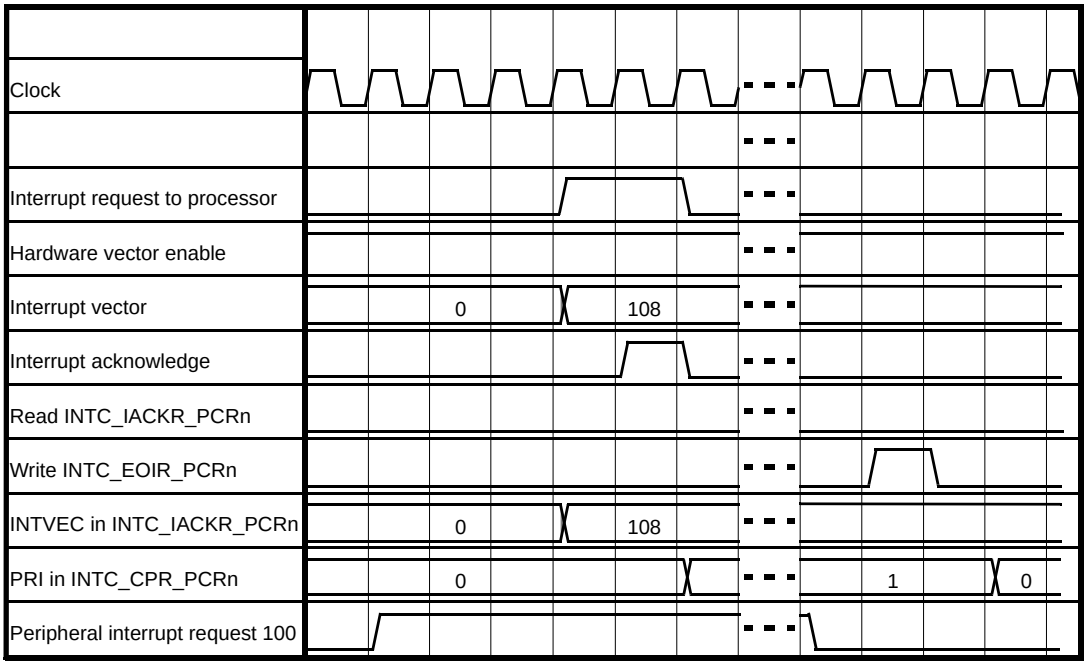


Figure 190. Hardware Vector Mode Handshaking Timing Diagram

19.10 Initialization/Application Information

19.10.1 Initialization flow

After exiting reset, all of the PRI*n* fields in INTC priority select registers (INTC_PSR0–INTC_PSR278) will be zero, and PRI in INTC current priority register (INTC_CPR_PRC0 and INTC_CPR_PRC1) 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_PRC0,VTES_PRC1,HVEN_PRC0 and HVEN_PRC1 in INTC_MCR
configure VTBA_PRCn in INTC_IACKR_PRCn
raise the PRIn fields and set the PRC_SELx fields to the desired
processor in INTC_PSRn_n
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR_PRCn to zero
enable processor(s) recognition of interrupts
```

19.10.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

19.10.2.1 Software vector mode

```
interrupt_exception_handler:
code to save SRR0 and SRR1

lisr3,hi(INTC_IACKR_PRCx)# form INTC_IACKR_PRCx address
orir3,r3,lo(INTC_IACKR_PRCx)
lwzr3,0x0(r3)# load INTC_IACKR_PRCx, which clears request to processor
lwzr3,0x0(r3)# load address of ISR from vector table

code to enable processor recognition of interrupts and save context
required by EABI

mtlrr3     # move INTC_IACKR_PRCx contents into link register
blrl       # branch to ISR; link register updated with epilog
           # address

epilog:
lisr3,hi(INTC_EOIR_PRCx)# form INTC_EOIR_PRC0 address
orir3,r3,lo(INTC_EOIR_PRCx)
li r4,0x0  # form 0 to write to INTC_EOIR_PRCx
stwr4,0x0(r3)# store to INTC_EOIR_PRCx, informing INTC to lower priority

code to restore context required by EABI and disable processor recognition
of interrupts
code to restore SRR0 and SRR1

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
```

19.10.2.2 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_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` 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

wrteei1    # 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
lisr3,INTC_EOIR_PRCn@ha# form adjusted upper half of INTC_EOIR_PRCn address
li r4,0x0  # form 0 to write to INTC_EOIR_PRCn
wrteei0    # disable processor recognition of interrupts
stwr4,INTC_EOIR_PRCn@l(r3)# store to INTC_EOIR_PRCn, 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
```

19.10.3 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_PRC0 or INTC_CPR_PRC1) having a value of 0. The RTOS executes the tasks according to whatever priority scheme 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_PRC*n* priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR_PRC*n* priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR_PRC*n* 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_PRC*n* while the shared resource is being accessed.An ISR whose PRI*n* in INTC priority select registers (INTC_PSR0–INTC_PSR278) has a value of 0 does not cause an interrupt request to the selected 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 causes it to remain negated, which consequently also does not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR does not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISROrder 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 184* shows the order of execution of both ISRs with different priorities and the same priority.

Table 184. 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

Table 184. 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	
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR_PRCn.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR_PRCn.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR_PRCn.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR_PRCn.						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.

19.10.4 Priority Ceiling protocol

19.10.4.1 Elevating priority

The PRI field in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) 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 all share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR_PRC*n* to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC_CPR_PRC*n* 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.

19.10.4.2 Ensuring Coherency

19.10.4.2.1 Interrupt with Blocked Priority

Non-coherent accesses to a shared resource can occur. As an example, ISR1 and ISR2 both share a resource. ISR1 has a lower priority, therefore it executes and then writes the new PRI value to the current priority register (INTC_CPR_PRC*n*). The next instruction writes a value to a shared coherent data block.

If INTC asserts the ISR2 interrupt request to the processor just before or at the same time as the first ISR1 write, it is possible for both the ISR1 and ISR2 writes to execute while the processor responds to the INTC request, discards the transactions, and flushes the processing pipeline. However, ISR2 cannot access the data block coherently because the data block is now corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corrupting a coherent data block, use the following code to modify the PRI in INTC_CPR_PRC*n*. Interrupts must be disabled before executing the following GetResource code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

19.10.4.2.2 Raised Priority Preserved

Before the instruction after the GetResource system service executes, all pending transactions have completed. These pending transactions can include an ISR for a peripheral or software settable interrupt request whose priority was equal to or lower than the raised priority. Also, during the epilog of the interrupt exception handler for this preempting ISR, the raised priority has been restored from the LIFO to PRI in INTC_CPR. The shared coherent data block now can be accessed coherently. *Figure 191* shows the timing diagram for this scenario, and *Table 185* explains the events. The example is for software vector mode, but except for the method of retrieving the vector and acknowledging the interrupt request to the processor, hardware vector mode is identical.

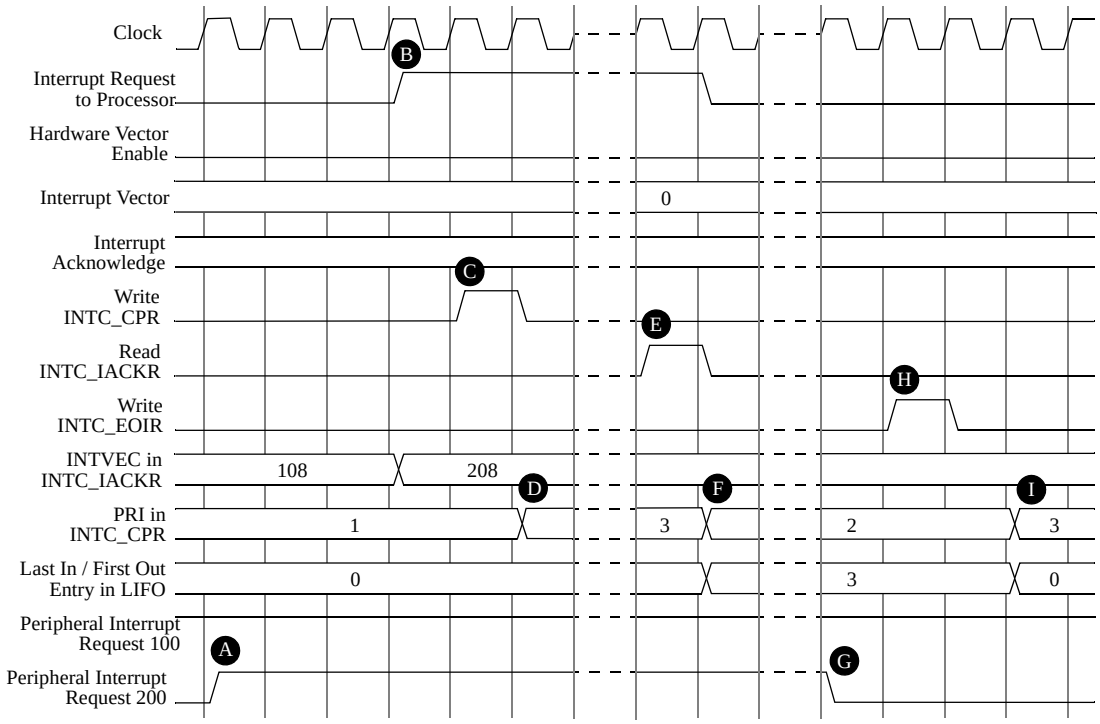


Figure 191. Raised Priority Preserved Timing Diagram

Table 185. Raised Priority Preserved Events

Event	Description
A	Peripheral interrupt request 200 asserts during execution of ISR108 running at priority 1.
B	Interrupt request to processor asserts. INTVEC in INTC_IACKR updates with vector for that peripheral interrupt request.
C	ISR108 writes to INTC_CPR to raise priority to 3 before accessing shared coherent data block.
D	PRI in INTC_CPR now at 3, reflecting the write. This write, just before accessing data block, is the last instruction the processor executes before being interrupted.
E	Interrupt exception handler prolog acknowledges interrupt by reading INTC_IACKR.
F	PRI of 3 pushed onto LIFO. PRI in INTC_CPR updates to 2, the priority of ISR208.
G	ISR208 clears its flag bit, deasserting its peripheral interrupt request.
H	Interrupt exception handler epilog writes to INTC_EOIR.
I	LIFO pops 3, restoring the raised priority onto PRI in INTC_CPR. Next value to pop from LIFO is the priority from before peripheral interrupt request 100 interrupted. ISR108 now can access data block coherently after interrupt exception handler executes rfi instruction.

19.10.5 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.

19.10.6 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.

19.10.6.1 Scheduling a Lower Priority Portion of an ISR

A portion of an ISR needs to be executed at the PRI*n* value in INTC priority select registers (INTC_PSR0–INTC_PSR278), which becomes the PRI value in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) with the interrupt acknowledge. The ISR, however, can have a portion of it which does not need to be executed at this higher priority. Therefore, executing this later portion that does not need to be executed at this higher priority can prevent the execution of ISRs that do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). Writing a 1 to SET*n* causes a software settable interrupt request. This software settable interrupt request usually has a lower PRI*n* value in the INTC_PSR*n*, and therefore does 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.

19.10.6.2 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.

19.10.7 Lowering Priority Within an ISR

In implementations without the software-settable interrupt requests in the INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7), one way — besides scheduling a task through an RTOS — to prevent preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities is to lower the current priority (see *Section 19.10.6.1: Scheduling a Lower Priority Portion of an ISR*). However, the INTC has a LIFO whose depth is determined by the number of priorities.

Note: Lowering the PRI value in either INTC_CPR_PRC0 or INTC_CPR_PRC1 within an ISR to below the ISR's corresponding PRI value in INTC_PSR0–INTC_PSR278 allows more preemptions than the LIFO depth can support.

Therefore, through its use of the LIFO, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

19.10.8 Negating an Interrupt Request Outside of its ISR

19.10.8.1 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.

19.10.8.2 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.

19.10.8.3 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_PSR276-278) 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 19.9.3.1.2: 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*.

19.10.9 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_PRCn
load INTC_CPR_PRCn, 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_PRCn
load INTC_IACKR_PRCn
if stacked PRI values are not depleted, branch to push_lifo
```

Note: Reading the INTC_IACKR_PRCn acknowledges the interrupt request to the processor and updates the INTC_CPR_PRCn[PRI] with the priority of the preempting interrupt request. If the processor recognition of interrupts is disabled during the LIFO restoration, interrupt requests to the processor can go undetected. However, since the peripheral or software settable interrupt requests are not cleared, the peripheral interrupt request to the processor re-asserts when INTC_CPR_PRCn[PRI] is lower than the priorities of those peripheral or software settable interrupt requests.

20 Crossbar Switch (XBAR)

20.1 Features

The following summarizes the device-specific implementation of the crossbar switch:

- Eight master ports
  - e200z4d Instruction port
  - e200z4d Data port
  - e200z0h Instruction port
  - e200z0h Data port
  - eDMA
  - FEC
  - FlexRay
  - CSE
- Five slave ports:
  - Flash controller supports 2 slave ports
  - PRAM controller supports 2 slave ports
  - PBRIDGE

Table 186. Master/slave mappings

XBAR Module	XBAR Port		Master ID
	Type	XBAR Port number	
e200z4d ifetch	Master	m0	0000
e200z4d dfetch	Master	m1	
e200z4d Nexus3+			1000
eDMA	Master	m2	0010
e200z0h ifetch	Master	m3	0001
e200z0h dfetch	Master	m4	
e200z0h Nexus3+			1001
FEC	Master	m5	0101
FlexRay	Master	m6	0110
Crypto XBAR_IF (CSE)	Master	m7	0111
Flash port dedicated to e200z4d ifetch	Slave	s0	—
Flash port dedicated to everything else	Slave	s1	—
PRAM Controller 0	Slave	s2	—
PRAM Controller 1	Slave	s3	—
PBRIDGE	Slave	s7	—

Note: Beware that the MPU module only reads the last three bits of a Master ID reference. Therefore Nexus masters (Master ID 8 and 9)" share the same permissions as cores Z4 and Z0 (masters 0

20.2 Introduction

20.2.1 Overview

This section provides an overview of the crossbar switch (XBAR). The purpose of the XBAR is to concurrently support up to 5 simultaneous connections between master ports and slave ports. The XBAR supports a 32-bit address bus width and 64-bit data bus width at all master and slave ports. Only a single data bus width is supported throughout the design, thus, all master and slave ports have the same data bus width.

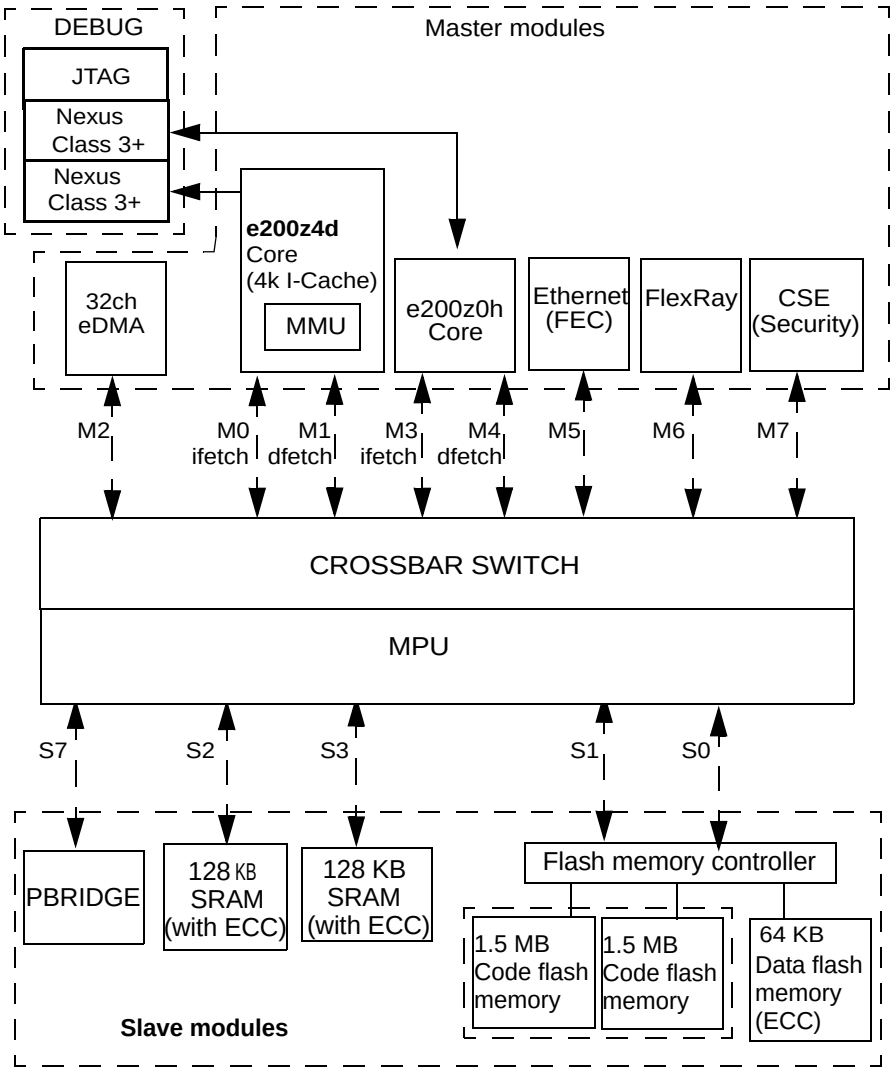


Figure 192. XBAR block diagram

20.2.2 Features

The XBAR has the ability to gain control of all the slave ports and prevent any masters from making accesses to the slave ports. This feature is useful when the user wishes to turn off the clocks to the system and needs to ensure that no bus activity will be interrupted.

The XBAR can put each slave port into a low power park mode so that slave port will not dissipate any power transitioning address, control or data signals when not being actively accessed by a master port.

Each slave port can also support multiple master priority schemes. Each slave port has a hardware input which selects the master priority scheme so the user can dynamically change master priority levels on a slave port by slave port basis.

The XBAR allows concurrent accesses between unique master and unique slave ports with no contention. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will be stalled until the higher priority master completes its transactions.

20.2.3 Limitations

The XBAR routes bus transactions initiated on the master ports to the appropriate slave ports. There is no provision included to route transactions initiated on the slave ports to other slave ports or to master ports. Simply put, the slave ports do not support the bus request/bus grant protocol, the XBAR assumes it is the sole master of each slave port.

20.2.4 General operation

When a master makes an access to the XBAR the access will be immediately taken by the XBAR. If the targeted slave port of the access is available then the access will be presented on the slave port. If the slave port is parked on the master initiating the access then there will be a zero clock delay through the crossbar for that access. If the targeted slave port of the access is busy or parked on a different master port the requesting master will see wait states inserted until the targeted slave port can service the master’s request. The latency in servicing the request will depend on each master’s priority level and the responding peripheral’s access time.

Since the XBAR appears to be just another slave to the master device, the master device will have no knowledge of whether or not it actually owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, wait states will be inserted until the slave port becomes available.

A master will be given control of the targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has an outstanding request to one slave port that has a long response time, has a pending access to a different slave port, and a lower priority master is also making a request to the same slave port as the pending access of the higher priority master.

After the 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 or by leaving that slave port for its next access. The master could also lose control of the slave port if another higher priority master makes a request to the slave port; however, if the master is running a locked or fixed length burst transfer it will retain control of the slave port until that transfer is

completed. Based on the AULB bit in the MGPCR (Master General Purpose Control Register) the master will either retain control of the slave port when doing undefined length incrementing burst transfers or will lose the bus to a higher priority master.

The XBAR will terminate all master IDLE transfers (as opposed to allowing the termination to come from one of the slave busses). Additionally, when no master is requesting access to a slave port the XBAR will drive IDLE transfers onto the slave bus, even though a default master may be granted access to the slave port.

When a slave bus is being IDLEd by the XBAR it can park the slave port on the master port indicated by the PARK bits in the SGPCR (Slave General Purpose Control Register). This can be done in an attempt to save the initial clock of arbitration delay that would otherwise be seen if the master had to arbitrate to gain control of the slave port. The slave port can also be put into low power park mode in attempt to save power.

20.3 XBAR registers

This section provides information on XBAR registers.

20.3.1 Register summary

There are 2 registers that reside in each slave port of the XBAR and one register that resides in each master port of the XBAR. These registers can only be read or written to via a 32-bit access in supervisor mode.

Each SGPCRn register includes a Read Only (RO) bit to protect both registers associated with that slave port. Once set, any attempt to write to the affected slave port registers will result in an IVOR1 exception. The RO bit setting remains in force until the next reset.

The memory map for the XBAR registers is shown in *Table 187*. All registers are 32 bits in size. All address offsets not explicitly mentioned in the table are reserved.

Table 187. XBAR memory map

Base address: 0xFFF0_4000		
Address offset	**Register**	**Location**
0x000	Master Priority Register for Slave port 0 (MPR0)	*on page 479*
0x010	General Purpose Control Register for Slave port 0 (SGPCR0)	*on page 480*
0x100	Master Priority Register for Slave port 1 (MPR1)	*on page 479*
0x110	General Purpose Control Register for Slave port 1 (SGPCR1)	*on page 480*
0x200	Master Priority Register for Slave port 2 (MPR2)	*on page 479*
0x210	General Purpose Control Register for Slave port 2 (SGPCR2)	*on page 480*
0x300	Master Priority Register for Slave port 3 (MPR3)	*on page 479*
0x310	General Purpose Control Register for Slave port 3 (SGPCR3)	*on page 480*
0x700	Master Priority Register for Slave port 7 (MPR7)	*on page 479*
0x710	General Purpose Control Register for Slave port 7 (SGPCR7)	*on page 480*
0x800	General Purpose Control Register for Master port 0 (MGPCR0)	*on page 482*
0x900	General Purpose Control Register for Master port 1 (MGPCR1)	*on page 482*

Table 187. XBAR memory map (continued)

Base address: 0xFFF0_4000		
Address offset	**Register**	**Location**
0xA00	General Purpose Control Register for Master port 2 (MGPCR2)	*on page 482*
0xB00	General Purpose Control Register for Master port 3 (MGPCR3)	*on page 482*
0xC00	General Purpose Control Register for Master port 4 (MGPCR4)	*on page 482*
0xD00	General Purpose Control Register for Master port 5 (MGPCR5)	*on page 482*
0xE00	General Purpose Control Register for Master port 6 (MGPCR6)	*on page 482*
0xF00	General Purpose Control Register for Master port 7 (MGPCR7)	*on page 482*

20.3.2 XBAR register descriptions

The following sections provide detailed descriptions of the various XBAR registers.

20.3.2.1 Master Priority Registers (MPRn)

The Master Priority Registers (MPRn) sets the priority of each master port on a per slave port basis and resides in each slave port.

Offset: 0x000 + n*100

Access: Supervisor Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MSTR_7			0	MSTR_6			0	MSTR_5			0	MSTR_4		
W																
Reset:	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MSTR_3			0	MSTR_2			0	MSTR_1			0	MSTR_0		
W																
Reset:	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figure 193. Master Priority Registers (MPRn)

Note: n represents the slave port number from 0 to 7.

Table 188. MPRn field descriptions

Field	Description
MSTR_7	**Master 7 Priority -** These bits set the arbitration priority for master port 7 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.
MSTR_6	**Master 6 Priority -** These bits set the arbitration priority for master port 6 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.

Table 188. MPRn field descriptions (continued)

Field	Description
MSTR_5	**Master 5 Priority -** These bits set the arbitration priority for master port 5 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.
MSTR_4	**Master 4 Priority -** These bits set the arbitration priority for master port 4 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.
MSTR_3	**Master 3 Priority -** These bits set the arbitration priority for master port 3 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.
MSTR_2	**Master 2 Priority -** These bits set the arbitration priority for master port 2 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.
MSTR_1	**Master 1 Priority -** These bits set the arbitration priority for master port 1 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.
MSTR_0	**Master 0 Priority -** These bits set the arbitration priority for master port 0 on the associated slave port. These bits are initialized by hardware reset. 000 This master has the highest priority when accessing the slave port. 111 This master has the lowest priority when accessing the slave port.

Note: No two master ports within the same MPR may be programmed with the same priority level. Any attempt to do so will result in a bus error and the MPR will not be updated.

20.3.2.2 Slave General Purpose Control Registers (SGPCRn)

The Slave General Purpose Control Registers (SGPCR) control several features of each slave port.

The Read Only (RO) bit will prevent any registers associated with this slave port from being written to once set. After RO is set, it can only be cleared by a reset.

The PCTL bits determine how the slave port will park when no master is actively making a request. The available options are:

- Park on a specific master defined by the PARK bits
- Park on the last master to use the slave port
- Park on no master and enter a low power park mode which will force all of the outputs of the slave port to inactive when no master is requesting an access.

The low power park feature can result in an overall power savings if a the slave port is not saturated; however, it will always result in an extra clock whenever the slave port is accessed by any master.

The PARK bits determine which master the slave will park on when no master is making an active request. Ensure that the slave is only parked on a master that exists in your microcontroller. If the slave is parked to master that does not exist, then undefined behavior will result.

Offset: 0x010 + n*100 Access: Supervisor Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO	0	0	0	0	0	0	0	HPE7	HPE6	HPE5	HPE4	HPE3	HPE2	HPE1	HPE0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Once the RO bit is written to a 1, only hardware reset will return it to a 0.

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ARB		0	0	PCTL		0	PARK		
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 194. Slave General Purpose Control Register n (SGPCRn)

Note: n represents the slave port number.

Table 189. SGPCRn field descriptions

Field	Description
RO	**Read Only -** This bit is used to force all of a slave port's registers to be read only. Once written to 1 it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0. 0 All this slave port's registers can be written. 1 All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
HPEx	**High Priority Enable -** These bits allow the respective master to assert a high priority, they do not enable the high priority access. In order to elevate priority for the cores or eDMA access, see *Section 20.4.2.1: Priority elevation*. 0 Priority elevation is gated on this slave port 1 Priority elevation is not gated on this slave port
ARB	**Arbitration Mode -** These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 00. 00 Fixed Priority. 01 Round Robin (rotating) Priority. 10 Reserved 11 Reserved
PCTL	**Parking Control -** These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 00. 00 When no master is making a request the arbiter will park the slave port on the master port defined by the PARK bit field. 01 When no master is making a request the arbiter will park the slave port on the last master to be in control of the slave port. 10 When no master is making a request the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11 Reserved

Table 189. SGPCRn field descriptions (continued)

Field	Description
PARK	**PARK -** These bits are used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset. The reset value is 000. 000 Park on Master Port 0 001 Park on Master Port 1 010 Park on Master Port 2 011 Park on Master Port 3 100 Park on Master Port 4 101 Park on Master Port 5 110 Park on Master Port 6 111 Park on Master Port 7

20.3.2.3 Master General Purpose Control Registers (MGPCRn)

The MGPCRs contain a field, AULB (Arbitrate on Undefined Length Bursts), that determines whether (and when) or not the XBAR will arbitrate away the slave port the master owns when the master is performing undefined length burst accesses.

Offset: 0x800 + n*100 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
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	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	AULB		
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Note:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 195. Master General Purpose Control Register n (MGPCRn)

Note: n represents the master port number from 0 to 7.

Table 190. MGPCRn field descriptions

Field	Description
AULB	**Arbitrate on Undefined Length Bursts -** These bits are used to select the arbitration policy during undefined length bursts by this master. These bits are initialized by hardware reset. The reset value is 000. 000 No arbitration will be allowed during an undefined length burst. 001 Arbitration will be allowed at any time during an undefined length burst. 010 Arbitration will be allowed after four beats of an undefined length burst. 011 Arbitration will be allowed after eight beats of an undefined length burst. 100 Arbitration will be allowed after 16 beats of an undefined length burst. 101 Reserved 110 Reserved 111 Reserved

The MGPCR can only be accessed in supervisor mode with 32-bit accesses.

20.3.3 Coherency

Since the content of the registers has a real time effect on the operation of the XBAR it is important for the user to understand that any register modifications take effect as soon as the register is written.

The exception to this rule are the MGPCRn[AULB] bits. The effect of modifying these bits is only realized when the master runs an IDLE cycle where the new settings take effect. If the AULB bits in the MGPCR are written in between two burst accesses the new AULB encodings will not take effect until an IDLE cycle has been initiated by the master on that master port.

20.4 Function

This section describes in more detail the functionality of the XBAR.

20.4.1 Arbitration

The XBAR supports two arbitration schemes:

- A fixed-priority comparison algorithm
- A round-robin fairness algorithm

The arbitration scheme is independently programmable for each slave port.

20.4.1.1 Arbitration During Undefined Length Bursts

Arbitration during an undefined length burst are defined by the current master's MGPCR AULB field setting. When a defined length is imposed on the burst via the AULB bits the undefined length burst will be treated as a single or series of single back to back fixed length burst accesses.

Example: A master runs an undefined length burst and the AULB bits in the MGPCR indicate arbitration will occur after the fourth beat of the burst. The master runs two sequential beats and then starts what will be an 12 beat undefined length burst access to a new address within the same slave port region as the previous access. The XBAR will not allow an arbitration point until the fourth overall access (second beat of the second burst). At that point all remaining accesses will be open for arbitration until the master loses control of the slave port.

Assume the master loses control of the slave port after the fifth beat of the second burst. Once the master regains control of the slave port no arbitration point will be available until after the master has run four more beats of its burst. After the fourth beat of the (now continued) burst (ninth beat of the second burst from the master's perspective) is taken all beats of the burst will once again be open for arbitration until the master loses control of the slave port.

Assume the master again loses control of the slave port on the fifth beat of the third (now continued) burst (10th beat of the second burst from the master's perspective). Once the master regains control of the slave port it will be allowed to complete its final two beats of its burst without facing arbitration.

Note: Fixed length burst accesses are not affected by the AULB bits. All fixed length burst accesses lock out arbitration until the last beat of the fixed length burst.

20.4.1.2 Fixed priority operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the MPR (Master Priority Register). If two masters both request access to a slave port the master with the highest priority in the selected priority register will gain control over the slave port.

Since the e200z0 instruction bus can make indefinitely long accesses on XBAR, there is a strong possibility of starving all other masters for flash memory access if e200z0 instruction bus is made the highest-priority bus master while it is fetching code. Extra care with respect to other expected transactions from other masters should be taken if the e200z0 instruction bus is made highest priority bus master.

Any time a master makes a request to a slave port the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port the new requesting master will be granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed length burst transfer or a locked transfer. In this case the new requesting master will have to wait until the end of the burst transfer or locked transfer before it will be granted control of the slave port. If the master is running an undefined length burst transfer the new requesting master must wait until an arbitration point for the undefined length burst transfer before it will be granted control of the slave port. Arbitration points for an undefined length burst are defined in the MGPCR for each master.

If the new requesting master's priority level is lower than that of the master that currently has control of the slave port the new requesting master will be forced to wait until the master that currently has control of the slave port either runs an IDLE cycle or runs a non IDLE cycle to a location other than the current slave port.

20.4.1.3 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master number.This relative priority is compared to the ID of the last master to perform a transfer on the slave bus. The highest priority requesting master will become owner of the slave bus as the next transfer boundary (accounting for locked and fixed-length burst transfers). Priority is based on how far ahead the ID of the requesting master is to the ID of the last master (ID is defined by master port number, not the **hmaster** field).

Once granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line will be granted access to the slave port at the next assertion of **sX_hready**, or possibly on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the XBAR is implemented with master ports 0, 1, 4 and 5. If the last master of the slave port was master 1, and master 0, 4 and 5 make simultaneous requests, they will be serviced in the order 4, 5 and then 0.

Parking may still be used in a round-robin mode, but will not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff will occur to the next master in line after one cycle of arbitration. If the slave port is put into low power park mode the round-robin pointer will be reset to point at master port 0, giving it the highest priority.

Each master port has an **mX_high_priority** input which can be enabled by writing the correct data to the SGPCR or ASGPCR. If a master's **mX_high_priority** input is enabled for a slave

port programmed for round-robin mode, that master can force the slave port into fixed priority mode by asserting its **mX_high_priority** input while making a request to that particular slave port. While that (or any enabled) master's **mX_high_priority** input is asserted while making an access attempt to that particular slave port, the slave port will remain in fixed priority mode. Once that (or any enabled) master's **mX_high_priority** input is negated, or the master no longer attempts to make accesses to that particular slave port, the slave port will revert back to round-robin priority mode and the pointer will be set on the last master to access the slave port.

20.4.2 Priority assignment

Each master port needs to be assigned a unique 3-Bit priority level. If an attempt is made to program multiple master ports with the same priority level within a register (MPR) the XBAR will respond with an error and the registers will not be updated.

20.4.2.1 Priority elevation

The XBAR has a hardware input per master port (**mX_high_priority**) which is used to temporarily elevate the master's priority level on all slave ports. When the master's **mX_high_priority** input is asserted the master port will automatically have higher priority than all other master ports that do not have their **mX_high_priority** input asserted regardless of the priority levels programmed in the MPR and AMPR. If multiple master ports have their **mX_high_priority** input asserted they will have higher priority than all master ports which do not have their **mX_high_priority** inputs asserted. The MPR or AMPR priority level (dependent on the state of **sX_ampr_sel**) will determine which master port that has its **mX_high_priority** input asserted has the highest priority on a slave port by slave port basis.

This functionality is useful because it allows the user to automatically elevate a master port's priority level throughout the XBAR in order to quickly perform temporary tasks such as servicing interrupts.

Please note that the HPEx bits must be set in the SGPCR or ASGPCR in the slave port in order for the **mX_high_priority** inputs to be received by the slave port.

Priority elevation is only valid for e200z0, e200z4, and eDMA masters.

For priority elevation to work:

1. Write SYSCTL field in HID1 and set either EE or ME bit. This activates priority elevation.
2. Gate the priority elevation by writing respective HPEx bit within the ISR. Priority should now be elevated for access within the ISR
3. Set HPEx from slave port register that you are wanting elevate within an ISR.

20.4.3 Master Port Functionality

20.4.3.1 General

Each master port consists of two decoders, a capture unit, a register slice, a mux and a small state machine.

The first decoder is used to decode the **mX_hsel_slv** and control signals coming directly from the master, telling the state machine where the master's next access will be and if it is in fact a legal access. The second decoder receives its input from the capture unit, so it may be looking directly at the signals coming from the master or it may be looking at captured

signals coming from the master, depending entirely on the state of the targeted slave port. The second decoder is then used to generate the access requests that go to the slave ports.

The capture unit is used to capture the address and control information coming from the master in the event that the targeted slave port cannot immediately service the master. The capture unit is controlled by outputs from the state machine which tell it to either pass through the original master signals or the captured signals.

The register slice contains the registers associated with the specific master port. The registers have a quasi-IP bus interface at this level for reads and writes and the outputs feed directly into the state machine.

The mux is used simply to select which slave's read data is sent back to the master. The mux is controlled by the state machine.

The state machine controls all aspects of the master port. It knows which slave port the master wants to make a request to and controls when that request is made. It also has knowledge of each slave port, knowing whether or not the slave port is ready to accept an access from the master port. This will determine whether or not the master may immediately have its request taken by the slave port or whether the master port will have to capture the master's request and queue it at the slave port boundary.

20.4.3.2 Master Port Decoders

The decoders are very simple as they ensure an access request is allowed to be made and that the slave port targeted is actually present in the design. The decoders feeding the state machine are always enabled. The decoders that select the slave are enabled only when the master port controlling state machine wants to make a request to a slave port. This is necessary so that if a master port is making an access to a slave port and is being wait stated, and its next access is to a different slave port, the request to the second slave port can be held off until the access to the first slave port is terminated.

The decoders also output a "hole decode" or illegal access signal which tells the state machine that the master is trying to access a slave port that does not exist.

20.4.3.3 Master Port Capture Unit

The capture unit simply captures the state of the master's address and control signals if the XBAR cannot immediately pass the master's request through to the proper slave port. The capture unit consists of a set of flops and a mux which selects either the asynchronous path from address and control or the flopped (captured) address and control information.

20.4.3.4 Master Port Registers

The registers in the master port are only those registers associated with this particular master port. The read and write interface for the registers is a quasi-IP bus interface. It is not a full IP bus interface at this level because not all the IP bus signals are routed this deep in the design.

There is a register control block at the same level of the master port and slave port instantiations in the XBAR. This control block ensures that all accesses are 32-bit supervisor accesses before passing them on to the master ports.

The register outputs are connected directly to the state machine.

20.4.3.5 Master Port State Machine

20.4.3.5.1 Master Port State Machine States

The master side state machine's main function is to monitor the activities of the master port. The state machine has six states: **busy**, **idle**, **waiting**, **stalled**, **steady state**, **first cycle error response** and **second cycle error response**.

The **busy** state is used when the master runs a BUSY cycle to the master port. The master port maintains its request to the slave port if it currently owns the slave port; however, if it loses control of the slave port it will no longer maintain its request. If the master port loses control of the slave port it will not be allowed to make another request to the slave port until it runs a NSEQ or SEQ cycle.

The **idle** state is used when the master runs a valid IDLE cycle to the master port. The master port makes no requests to the slave ports (disables the slave port decoder) and terminates the IDLE cycle.

The **waiting** state is used when the **hsel** signal is negated to the master port, indicating the master is running valid cycles to a local slave other than the XBAR. In this case the max disables the slave port decoder and holds **hresp** and **hready** negated.

The **stalled** state is used when the master makes a request to a slave port that is not immediately ready to receive the request. In this case the state machine will direct the capture unit to send out the captured address and control signals and will enable the slave port decoder to indicate a pending request to the appropriate slave port.

The **steady state** state is used when the master port and slave port are in fully asynchronous mode, making the XBAR completely transparent in the access. The state machine selects the appropriate slave's **hresp, hready** and **hrdata** to pass back to the master.

The **first cycle error response** and **second cycle error response** states are self explanatory. The XBAR will respond with an error response to the master if the master tries to access an unimplemented memory location through the XBAR (that is, a slave port that does not exist).

20.4.3.5.2 Master Port State Machine Slave Swapping

The design of the master side state machine is fairly straightforward. The one real decision to be made is how to handle the master moving from one slave port access to another slave port access. The approach that was taken is to minimize or eliminate when possible any "bubbles" that would be inserted into the access due to switching slave ports.

The state machine will not allow the master to request access to another slave port until the current access being made is terminated. This prevents a single master from owning two slave ports at the same time (the slave port it is currently accessing and the slave port it wishes to access next).

The state machine also maintains watch on the slave port the master is accessing as well as the slave port the master wishes to switch to. If the new slave port is parked on the master then the master will be able to make the switch without incurring any delays. The termination of the current access will also act as the launch of the new access on the new slave port. If the new slave port is not parked on the master then the master will incur a minimum one clock delay before it can launch its access on the new slave port.

This is the same for switching from the **busy, idle** or **waiting** state to actively accessing a slave port. If the slave port is parked on the master the state machine will go to the **steady state** state and the access will begin immediately. If the slave port is not parked on the master

(serving another master, parked on another master or in low power park mode) then the state machine will transition to the **stalled** state and at least a one clock penalty will be paid.

20.4.4 Slave Port Functionality

20.4.4.1 General

Each slave port consists of a register slice, a bank of muxes and a state machine.

The register slice contains the registers associated with the specific slave port. The registers have a quasi-IP bus interface at this level for reads and writes and the outputs feed directly into the state machine.

The muxes are a series of 8 to 1 muxes that take in all the address, control and write data information from each of the master ports and then pass the correct master's signals to the slave port. The state machine controls all the muxes.

The state machine is where the main slave port arbitration occurs, it decides which master is in control of the slave port and which master will be in control of the slave port in the next bus cycle.

20.4.4.2 Slave Port Muxes

The block diagram shows only one block for all the muxes. In reality that block instantiates many 8 to 1 muxes, one for each master-to-slave signal in fact. All the muxes are designed in an AND - OR fashion, so that if no master is selected the output of the muxes will be zero. (This is an important feature for low power park mode.)

The muxes also have an override signal which is used by the slave port to asynchronously force IDLE cycles onto the slave bus. When the state machine forces an IDLE cycle it zeros out **htrans** and **hmastlock**, making sure the slave bus sees a valid IDLE cycle being run by the XBAR.

The enable to the mux controlling **htrans** also contains an additional control signal from the state machine so that a NSEQ transaction can be forced. This is done any time the slave port switches masters to ensure that no IDLE-SEQ, BUSY-SEQ or NSEQ-SEQ transactions are seen on the slave port when they shouldn't be. If the state machine indicates to run both an IDLE and an NSEQ cycle, the IDLE directive will have priority.

Note: IDLE-SEQ is in fact an illegal access, but a possible scenario given the multi-master environment in the XBAR unless corrected by the XBAR.

20.4.4.3 Slave Port Registers

There is a register control block at the same level of the master port and slave port instantiations in the XBAR. This control block ensures that all accesses are 32-bit supervisor accesses before passing them on to the master and slave ports.

The registers in the slave port are only those registers associated with this particular slave port. The read and write interface for the registers is a quasi-IP bus interface. It is not a full IP bus interface at this level because not all the IP bus signals are routed this deep in the design.

The register outputs are connected directly to the slave state machine with the **sX_ampr_sel** input determining which priority register values, halt priority value, arbitration algorithm and parking control bits are passed to the state machine. The registers can be read from an unlimited number of times. The registers can only be written to as long as the RO bit is written

to 0 in the SGPCR, once it is written to a 1 only a hardware reset will allow the registers to be written again.

20.4.4.4 Slave Port State Machine

20.4.4.4.1 Slave Port State Machine States

At the heart of the slave port is the state machine. The state machine is simplicity itself, requiring only four states - **steady state**, **transition state, transition hold state** and **hold state**. Either the slave port is owned by the same master it was in the last clock cycle (either by active use or by parking), it is transitioning to a new master (either for active use or parking), it is transitioning to a new master during wait states or it is being held on the same master pending a transition to a new master.

20.4.4.4.2 Slave Port State Machine Arbitration

The real work in the state machine is determining which master port will be in control of the slave port in the next clock cycle, the arbitration. Each master is programmed with a fixed 3 bit priority level. A fourth priority bit is derived from the **mX_high_priority** inputs on the master ports, effectively making each master's priority level a 4 bit field with **mX_high_priority** being the MSB. The XBAR uses these bits in determining priority levels when programmed for fixed priority mode of operation or when one of the enabled **mX_high_priority** inputs is asserted.

Arbitration always occurs on a clock edge, but only occurs on edges when a change in mastership will not violate AHB-Lite protocols. Valid arbitrations points include any clock cycle in which **sX_hready** is asserted (provide the master is not performing a burst or locked cycle) and any wait state in which the master owning the bus indicates a transfer type of IDLE (provided the master is not performing a locked cycle).

Since arbitration can occur on every clock cycle the slave port masks off all master requests if the current master is performing a locked transfer or a protected burst transfer, guaranteeing that no matter how low its priority level it will be allowed to finish its locked or protected portion of a burst sequence.

20.4.4.4.3 Slave Port State Machine Master Handoff

The only times the slave port will switch masters when programmed for fixed priority mode of operation is when a higher priority master makes a request or when the current master is the highest priority and it gives up the slave port by either running and IDLE cycle to the slave port or running a valid access to a location other than the slave port.

If the current master loses control of the slave port because a higher priority master takes it away, the slave port will not incur any wasted cycles. The current master has its current cycle terminated by the slave port at the same time the new master's address and control information are recognized by the slave port. This appears as a seamless transition on the slave port.

If the current master is being wait-stated when the higher priority master makes its request, then the current master will be allowed to make one more transaction on the slave bus before giving it up to the new master. *Figure 196* illustrates the effect of a higher priority master taking control of the bus when the slave port is programmed for a fixed priority mode of operation.

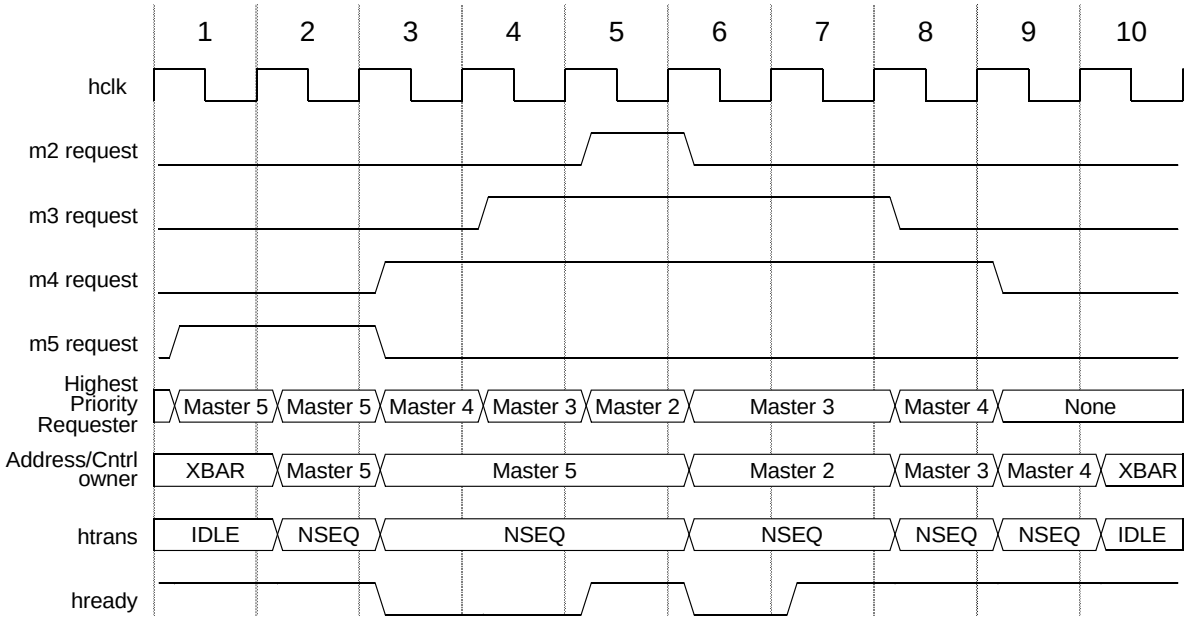


Figure 196. Low to high priority mastership change

If the current master is the highest priority master and it gives up the slave port by running an IDLE cycle or by running a valid cycle to another location other than the slave port the next highest priority master will gain control of the slave port. If the current access incurs any wait states then the transition will be seamless and no bandwidth will be lost; however, if the current transaction is terminated without wait states then one IDLE cycle will be forced onto the slave bus by the XBAR before the new master will be able to take control of the slave port. If no other master is requesting the bus then IDLE cycles will be run by the XBAR but no bandwidth will truly be lost since no master is making a request. *Figure 197* illustrates the effect of a higher priority master giving up control of the bus.

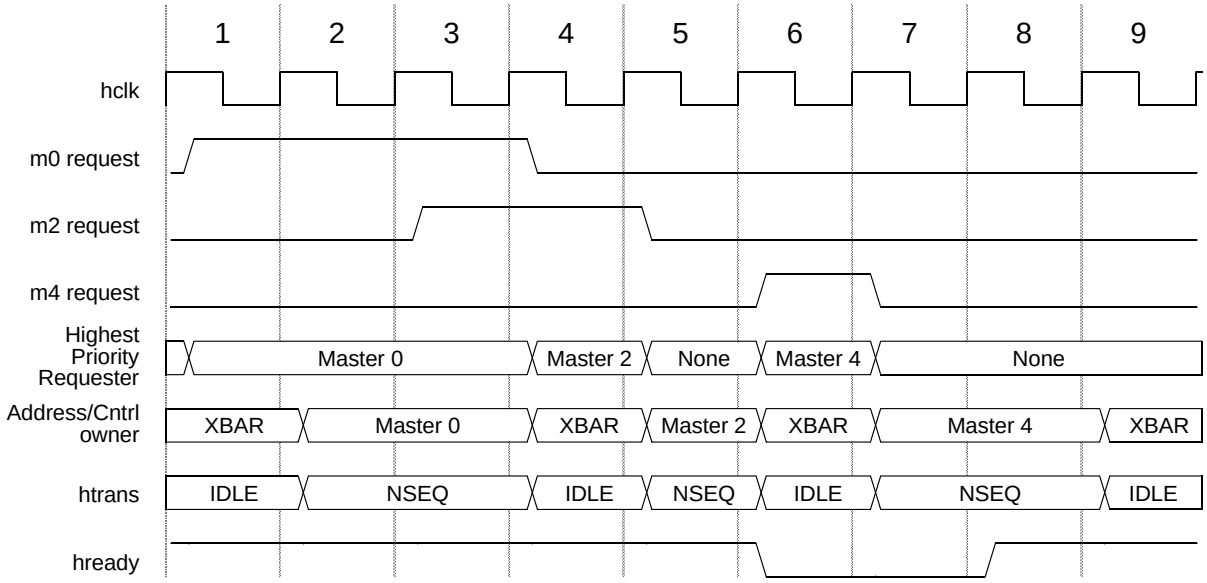


Figure 197. High to low priority mastership change

When the slave port is programmed for round-robin mode of arbitration then the slave port will switch masters any time there is more than one master actively making a request to the slave port. This will happen because any master other than the one which presently owns the bus will be considered to have higher priority. *Figure 198* shows an example of round-robin mode of operation.

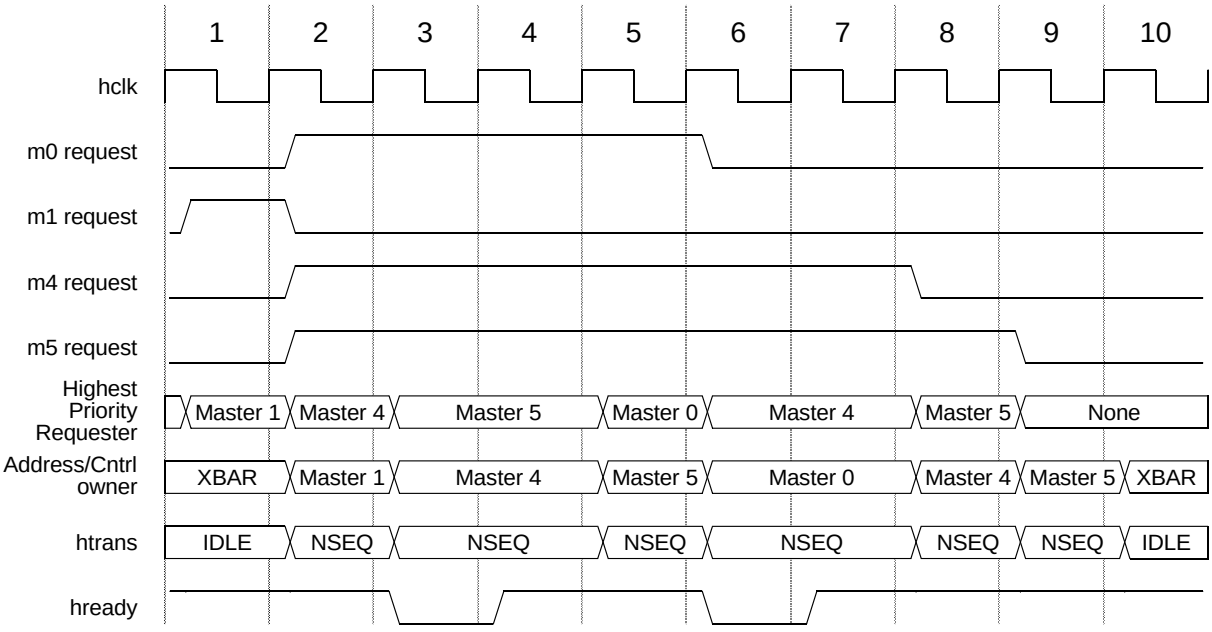


Figure 198. Round-robin mastership change

20.4.4.4.4 Slave Port State Machine Parking

If no master is currently making a request to the slave port then the slave port will be parked. It will park in one of four places, dictated by the PCTL and PARK bits in the GPCR or AGPCR (depending on the state of the **sX_ampr_sel**) and the locked state of the last master to access it.

If the last master to access the slave port ran a locked cycle and continues to run locked cycles even after leaving the slave port the slave port will park on that master without regard to the bit settings in the GPCR and without regard to pending requests from other masters. This is done so a master can run a locked transfer to the slave port, leave it, and return to it and be guaranteed that no other master has had access to it (provided the master maintains all transfers are locked transfers). If locking is not an issue for parking the GPCR bits will dictate the parking method.

If the PCTL bits are set for “low power park” mode then the slave port will enter low power park mode. It will not recognize any master as being in control of it and it will not select any master’s signals to pass through to the slave bus. In this case all slave bus activity will effectively halt because all slave bus signals being driven from the XBAR will be 0. This of course can save quite a bit of power if the slave port will not be in use for some time. The down side is that when a master does make a request to the slave port it will be delayed by one clock since it will have to arbitrate to acquire ownership of the slave port.

If the PCTL bits are set to “park on last” mode then the slave port will park on the last master to access it, passing all that masters signals through to the slave bus. The XBAR will asynchronously force **htrans[1:0]**, **hmaster[3:0]**, **hburst[2:0]** and **hmastlock** to 0 for all access that the master does not run to the slave port. When that master access the slave port again it will not pay any arbitration penalty; however, if any other master wishes to access the slave port a one clock arbitration penalty will be imposed.

If the PCTL bits are set to “use PARK/APARK” mode then the slave port will park on the master designated by the PARK bits. The behavior here is the same as for the “park on last” mode with the exception that a specific master will be parked on instead of the last master to access the slave port. If the master designated by the PARK bits tries to access the slave port it will not pay an arbitration penalty while any other master will pay a one clock penalty. *Figure 199* illustrates parking on a specific master.

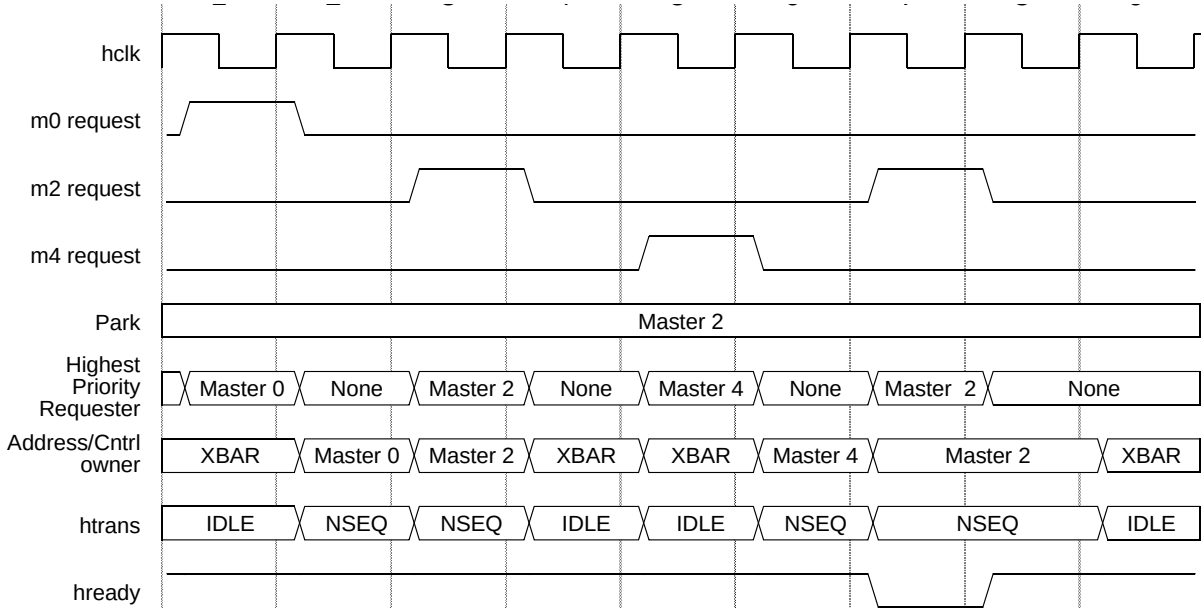


Figure 199. Parking on a specific master

Figure 200 illustrates parking on the last master. Note that in cycle 6 simultaneous requests are made by master 2 and master 4. Although master 2 has higher priority, the slave bus is parked on master 4 so master 4's access will be taken first. The slave port parks on master 2 once it has given control to master 2. This same situation can occur when parking on a specific master as well.

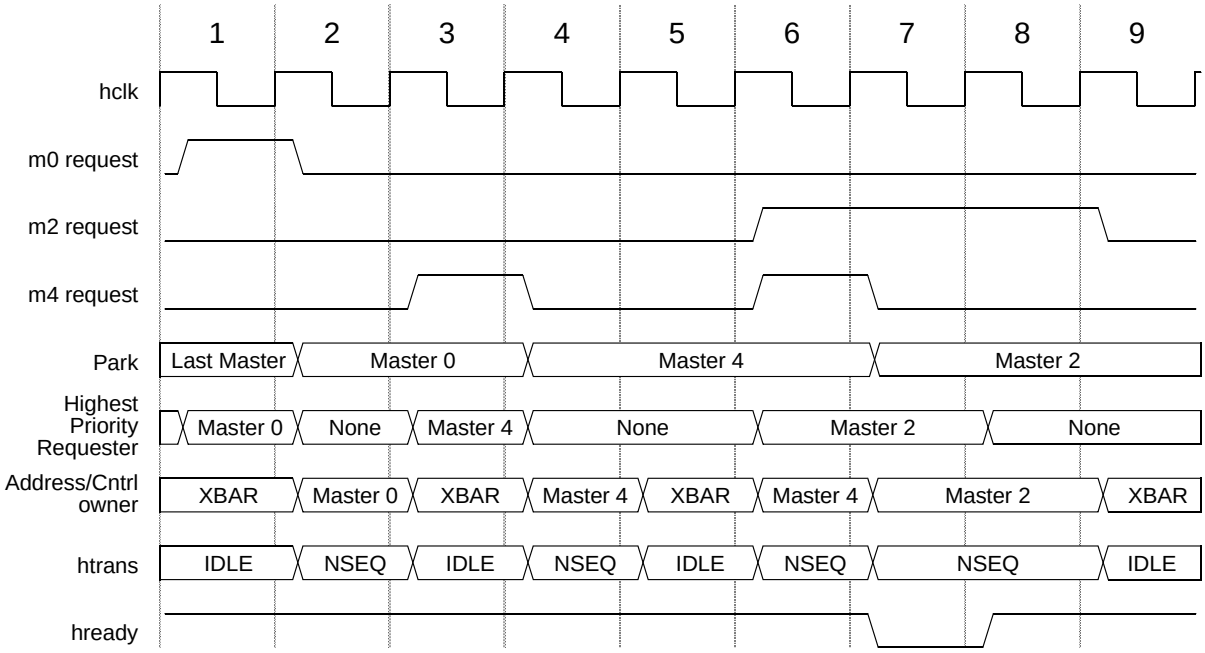


Figure 200. Parking on last master

20.4.4.4.5 Slave Port State Machine Halt Mode

If the **max_halt_request** input is asserted the slave port will eventually halt all slave bus activity and go into halt mode, which is almost identical to low power park mode. The HLP bit in the GPCR controls the priority level of the **max_halt_request** in the arbitration algorithm. If the HLP bit is cleared then the **max_halt_request** will have the highest priority of any master and will gain control of the slave port at the next arbitration point (most likely the next bus cycle, unless the current master is running a locked or fixed length burst transfer). If the HLP bit is set then the slave port will wait until no masters are actively making requests before moving to halt mode.

Regardless of the state of the HLP bit, once the slave port has gone into halt mode as a result of **max_halt_request** being asserted, it will remain in halt mode until **max_halt_request** is negated, regardless of the priority level of any masters that may make requests.

In halt mode no master is selected to own the slave port so all the outputs of the slave port are set to 0.

20.5 Initialization/Application Information

No initialization is required by or for the XBAR. Hardware reset ensures all the register bits used by the XBAR are properly initialized.

20.6 Interface

This section provides information on the XBAR interface.

20.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate in parallel with multiple slaves. In order to maximize data throughput it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR will stall the masters or insert bubbles on the slave side.

20.6.2 Master Ports

Master accesses will receive one of four responses from the XBAR. They will either be ignored, terminated, taken, stalled or responded to with an error.

20.6.2.1 Ignored Accesses

A master access will be ignored if the **hsel** input of the XBAR is not asserted. The XBAR will respond to IDLE transfers when the **hsel** input is asserted but will not allow the access to pass through the XBAR.

20.6.2.2 Terminated Accesses

A master access will be terminated if the **hsel** input of the XBAR is asserted and the transfer type is IDLE. The XBAR will terminated the access and it will not be allowed to pass through the XBAR.

20.6.2.3 Taken Accesses

A master access will be taken if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case the XBAR will be completely transparent and the master's access will be immediately seen on the slave bus and no arbitration delays will be incurred.

20.6.2.4 Stalled Accesses

A master access will be stalled if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the access decodes to a slave port that is busy serving another master, parked on another master or is in low power park mode. The XBAR will indicate to the master that the address phase of the access has been taken but will then queue the access to the appropriate slave port to enter into arbitration for access to that slave port.

If the slave port is currently parked on another master or is in low power park mode and no other master is requesting access to the slave port then only one clock of arbitration will be incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters then the master will gain control over the slave port as soon as the data phase of the current access is completed (burst and locked transfers excluded). If the slave port is currently servicing another master of a higher priority then the master will gain control of the slave port once the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

20.6.2.5 Error Response Terminated Accesses

A master access will be responded to with an error if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the access decodes to a location not occupied by a slave port. This is the only time the XBAR will respond with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

20.6.3 Slave Ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. In order to do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR will force a bubble onto the slave bus when a master is actively making a request. This occurs when a higher priority master has control of the slave port and is running single clock (zero wait state) accesses while a lower priority master is stalled waiting for control of the slave port. When the higher priority master either leaves the slave port or runs an IDLE cycle to the slave port the XBAR will take control of the slave bus and run a single IDLE cycle before giving the slave port to the lower priority master that was waiting for control of the slave port.

The only other times the XBAR will have control of the slave port is when the XBAR is halting or when no masters are making access requests to the slave port and the XBAR is forced to either park the slave port on a specific master or put the slave port into low power park mode.

In most instances when the XBAR has control of the slave port it will indicate IDLE for the transfer type, negate all control signals and indicate ownership of the slave bus via the **hmaster** encoding of 4'b0000. One exception to this rule is when a master running locked cycles has left the slave port but continues to run locked cycles. In this case the XBAR will

control the slave port and will indicate IDLE for the transfer type but it will not affect any other signals.

Note: When a master runs a locked cycle through the XBAR, the master will be guaranteed ownership of all slave ports it accesses while running locked cycles for one cycle beyond when the master finishes running locked cycles.

21 Memory protection unit (MPU)

21.1 Introduction

The AMBA-AHB 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. This module is commonly included as part of the platform.

21.1.1 Overview

The MPU module provides the following capabilities:

- Support for 16 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 five slave AHB ports: flash controller, system RAM controller and IPS peripherals bus
    - Connections to the AHB address phase address and attributes
    - Typical location is immediately “downstream” of the platform’s crossbar switch
  - Connection to the IPS bus provides access to the MPU’s programming model

A simplified block diagram of the AHB_MPU module is shown in *Figure 201*. The AHB bus slave ports (s{0,1,2,3}_h*) are shown on the right side of the diagram, the region descriptor registers in the middle and the IPS bus interface (ips_*) on the left side. The evaluation macro contains the two magnitude comparators connected to the start and end address registers from each region descriptor (rgdn) as well as the combinational logic blocks to determine the region hit and the access protection error. For information on the details of the access evaluation macro, see *Section 21.3.1: Access evaluation macro*.

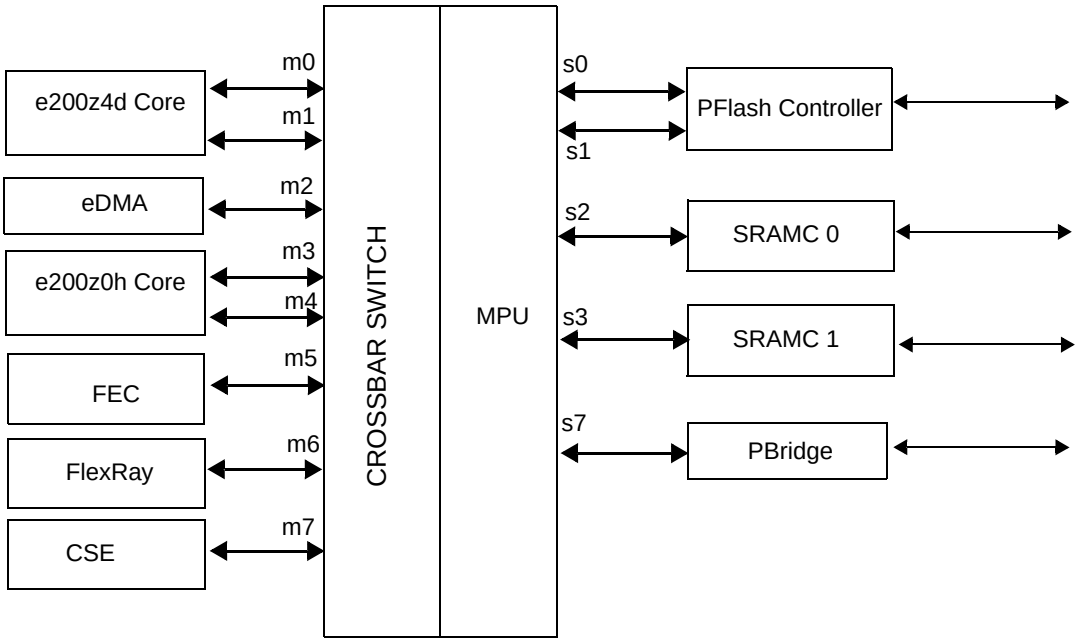


Figure 201. MPU block diagram

Note: This diagram refers to the XBAR port numbers and not the Master IDs. For more information, please refer to Table 187: XBAR memory map. Beware that the MPU module only reads the last three bits of a Master ID reference. Therefore, Nexus masters (Master ID 8 and 9) share the same permissions as cores Z4 and Z0 (Masters IDs 0 and 1).

21.1.2 Features

The Memory Protection Unit implements a two-dimensional hardware array of memory region descriptors and the crossbar slave AHB ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for 16 memory region descriptors, each 128 bits in size
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 Gbytes
  - Access control definitions:
    2 bus masters (processor cores) support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses.
    Reset of the bus masters (CSE, eDMA, etc.) {read, write}
  - 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 21.3.2: Putting it all together and AHB error terminations, for details and *Section 21.5: Application information*, for an example.

- Support for five AHB slave port connections: flash controller, system RAM controller and IPS peripherals bus
  - MPU hardware continuously monitors every AHB 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 AHB 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 AHB slave port, capture the last faulting address, attributes and “detail” information
- Global MPU enable/disable control bit provides a mechanism to easily load region descriptors during system startup or allow complete access rights during debug with the module disabled

21.1.3 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 IPS bus is used to access the MPU’s programming model and the memory protection functions are evaluated on a reference-by-reference basis using the addresses from the AHB system bus port(s).

Power dissipation is minimized when the MPU’s global enable/disable bit is cleared (MPU_CESR[VLD] = 0).

21.1.4 External signal description

The MPU module does not include any external interface. The MPU’s internal interfaces include an IPS 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 AHB slave ports.

21.2 Memory map and register description

The MPU module provides an IPS programming model mapped to an SPP-standard on-platform 16 Kbyte 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 16 region descriptors and up to 5 AHB slave ports, like flash controller, system RAM controller and IPS peripherals bus.

21.2.1 Memory map

The MPU programming model map is shown in *Table 191*.

Table 191. MPU memory map

Base address: 0xFFF1_1000		
Address offset	**Register**	**Location**
0x0000	MPU Control/Error Status Register (MPU_CESR)	*on page 502*
0x0004–0x000F	Reserved	
0x0010	MPU Error Address Register, Slave Port 0 (MPU_EAR0)	*on page 503*
0x0014	MPU Error Detail Register, Slave Port 0 (MPU_EDR0)	*on page 504*
0x0018	MPU Error Address Register, Slave Port 1 (MPU_EAR1)	*on page 503*
0X001C	MPU Error Detail Register, Slave Port 1 (MPU_EDR1)	*on page 504*
0x0020	MPU Error Address Register, Slave Port 2 (MPU_EAR2)	*on page 503*
0x0024	MPU Error Detail Register, Slave Port 2 (MPU_EDR2)	*on page 504*
0x0028	MPU Error Address Register, Slave Port 3 (MPU_EAR3)	*on page 503*
0x002C	MPU Error Detail Register, Slave Port 3 (MPU_EDR3)	*on page 503*
0x0030	MPU Error Address Register, Slave Port 4 (MPU_EAR4)	*on page 503*
0x0034	MPU Error Detail Register, Slave Port 4 (MPU_EDR4)	*on page 503*
0x0038–0x03FF		
0x0400	MPU Region Descriptor 0 (MPU_RGD0)	*on page 505*
0x0410	MPU Region Descriptor 1 (MPU_RGD1)	*on page 505*
0x0420	MPU Region Descriptor 2 (MPU_RGD2)	*on page 505*
0x0430	MPU Region Descriptor 3 (MPU_RGD3)	*on page 505*
0x0440	MPU Region Descriptor 4 (MPU_RGD4)	*on page 505*
0x0450	MPU Region Descriptor 5 (MPU_RGD5)	*on page 505*

Table 191. MPU memory map (continued)

Base address: 0xFFF1_1000		
Address offset	**Register**	**Location**
0x0460	MPU Region Descriptor 6 (MPU_RGD6)	*on page 505*
0x0470	MPU Region Descriptor 7 (MPU_RGD7)	*on page 505*
0x0480	MPU Region Descriptor 8 (MPU_RGD8)	*on page 505*
0x0490	MPU Region Descriptor 9 (MPU_RGD9)	*on page 505*
0x04A0	MPU Region Descriptor 10 (MPU_RGD10)	*on page 505*
0x04B0	MPU Region Descriptor 11 (MPU_RGD11)	*on page 505*
0x04C0	MPU Region Descriptor 12 (MPU_RGD12)	*on page 505*
0x04D0	MPU Region Descriptor 13 (MPU_RGD13)	*on page 505*
0x04E0	MPU Region Descriptor 14 (MPU_RGD14)	*on page 505*
0x04F0	MPU Region Descriptor 15 (MPU_RGD15)	*on page 505*
0x0500–0x07FF		
0x0800	MPU RGD Alternate Access Control 0 (MPU_RGDAAC0)	*on page 511*
0x0804	MPU RGD Alternate Access Control 1 (MPU_RGDAAC1)	*on page 511*
0x0808	MPU RGD Alternate Access Control 2 (MPU_RGDAAC2)	*on page 511*
0x080C	MPU RGD Alternate Access Control 3 (MPU_RGDAAC3)	*on page 511*
0x0810	MPU RGD Alternate Access Control 4 (MPU_RGDAAC4)	*on page 511*
0x0814	MPU RGD Alternate Access Control 5 (MPU_RGDAAC5)	*on page 511*
0x0818	MPU RGD Alternate Access Control 6 (MPU_RGDAAC6)	*on page 511*
0X081C	MPU RGD Alternate Access Control 7 (MPU_RGDAAC7)	*on page 511*
0X0820	MPU RGD Alternate Access Control 8 (MPU_RGDAAC8)	*on page 511*

Table 191. MPU memory map (continued)

Base address: 0xFFF1_1000		
Address offset	**Register**	**Location**
0X0824	MPU RGD Alternate Access Control 9 (MPU_RGDAAC9)	*on page 511*
0X0828	MPU RGD Alternate Access Control 10 (MPU_RGDAAC10)	*on page 511*
0X082C	MPU RGD Alternate Access Control 11 (MPU_RGDAAC11)	*on page 511*
0X0830	MPU RGD Alternate Access Control 12 (MPU_RGDAAC12)	*on page 511*
0X0834	MPU RGD Alternate Access Control 13 (MPU_RGDAAC13)	*on page 511*
0X0838	MPU RGD Alternate Access Control 14 (MPU_RGDAAC14)	*on page 511*
0X083C	MPU RGD Alternate Access Control 15 (MPU_RGDAAC15)	*on page 511*
0x0840–0x3FFF		

21.2.2 Register description

21.2.2.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset: 0x000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SPERR[7:0](1)								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	0	0	0	1

	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	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0

Figure 202. MPU Control/Error Status Register (MPU_CESR)

1. SPERR[7:5] are not used.

Table 192. MPU_CESR field descriptions

Field	Description
SPERR[7:0]	Slave Port n Error, where the slave port number matches the bit number SPERR[7] - Flash controller slave port SPERR[6] - Flash controller slave port SPERR[5] - System RAM controller slave port SPERR[4] - System RAM controller slave port SPERR[3] - IPS peripheral bus slave port 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. This field contains values of 0b0001–0b1000, depending on the device configuration.
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
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.

21.2.2.2 MPU Error Address Register, Slave Port n (MPU_EARn)

When the MPU detects an access error on slave port n, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDRn register at the same time. 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.

Offsets: 0x010–0x030 (5 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	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 203. MPU Error Address Register, Slave Port n(1) (MPU_EARn)

1. MPU_EAR4 register is for slave port 7.

Table 193. MPU_EARn field descriptions

Field	Description
EADDR	Error Address This field is the reference address from slave port n that generated the access error.

21.2.2.3 MPU Error Detail Register, Slave Port n (MPU_EDRn)

When the MPU detects an access error on slave port n, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU_EARn register at the same time. Note 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.

Offsets: 0x014–0x024 (5 registers) Access: Read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EPID								EMN				EATTR			ERW
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 204. MPU Error Detail Register, Slave Port n (MPU_EDRn)

Table 194. 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 Master ID 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).
ERW	Error Read/Write This field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

21.2.2.4 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.

21.2.2.4.1 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 21.2.2.4.4: MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)* for more information).

Offset: 0x400 + (16*n) + 0x0 (MPU_RGDn.Word0) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	SRTADDR[26:11]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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	0	0	0	0	0	0	0	0	0	0	0

Figure 205. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0)

Table 195. 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.

21.2.2.4.2 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 21.2.2.4.4: MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)* for more information).

Offset: 0x400 + (16 x n) + 0x4 (MPU_RGDn.Word1) Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	ENDADDR[26:11]															
Reset (n = 0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reset (n >0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENDADDR[10:0]											1	1	1	1	1
W																
Reset (n = 0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reset (n >0)	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 206. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1)

Table 196. 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.

21.2.2.4.3 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 AHB 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 21.2.2.4.4: MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)* for more information). Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor's valid bit.

Offset: 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(1)	M4WE(1)	M3PE(1)	M3SM(1)		M3UM(1)			M2PE	M2SM[1]
Reset (n = 0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Reset (n > 1)	0	0	0	0	0	0	0	0	0	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	M2SM[0]	M2UM(2)			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
Reset (n = 0)	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n > 1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 207. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2)

1. In SPC564Bx, there is no e200z0 core so masters with ID's 0011 and 0100 are not present. In this case, M3UM[2:0], M3SM[1:0], M3PE, M4WE, and M4RE bits are not valid.
2. M2UM[0] (eDMA) that corresponds to x (excute) is not valid.

Table 197. 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 197. 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[1:0]	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 197. 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.

21.2.2.4.4 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.

Offset: 0x400 + (16*n) + 0xc (MPU_RGDn.Word3) Access: R/W

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

Figure 208. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3)

Table 198. MPU_RGDn.Word3 field descriptions

Field	Description
PID[7:0]	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 21.3.1.1: 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

21.2.2.5 MPU Region Descriptor Alternate Access Control n (MPU_RGDAACn)

As noted in *Section 21.2.2.4.3: MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2)*, it is expected that since system software may adjust only the access controls within a region descriptor (MPU_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (Alternate Access Control n) as stores to these locations do not affect the descriptor's valid bit.

The memory address therefore provides an alternate location for updating MPU_RGDn.Word2.

Offset: 0x800 + (4*n) (MPU_RGDAACn) 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 (n = 0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Reset (n = 1)	0	0	0	0	0	0	0	0	0	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	M2SM[0]	M2UM			M1PE	M1SM			M1UM			M0PE	M0SM		M0UM[	
Reset (n = 0)	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n = 1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 209. MPU RGD Alternate Access Control n (MPU_RGDAACn)

Since the MPU_RGDAACn register is simply another memory mapping for MPU_RGDn.Word2, the field definitions shown in *Table 199* are identical to those presented in *Table 197*.

Table 199. 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.

Table 199. MPU_RGDAACn 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 199. MPU_RGDAACn 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.

21.3 Functional description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated AHB bus cycles.

21.3.1 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 210*, the access evaluation macro inputs the AHB system bus address phase signals (AHB_ap) and the contents of a region descriptor (RGDn) and performs two major functions: region hit determination (hit_b) and detection of an access protection violation (error).

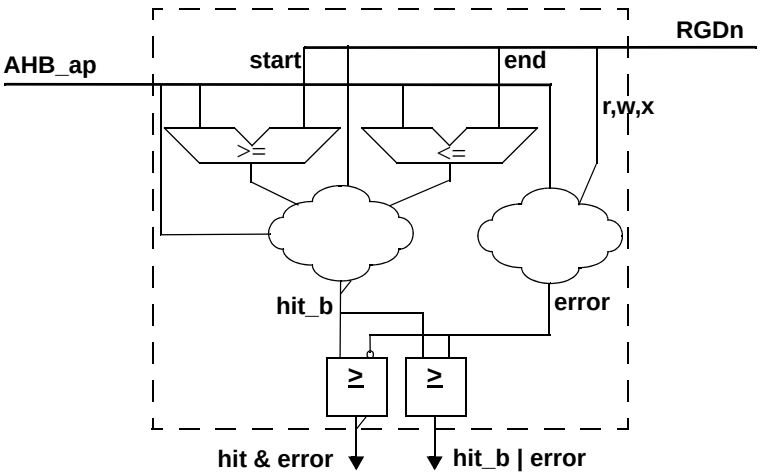


Figure 210. MPU access evaluation macro

Figure 210 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.

21.3.1.1 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.

21.3.1.2 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 200*.

Table 200. 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

Table 200. Protection violation definition (continued)

Description	Inputs			Output
	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection violation?
inst fetch read	—	—	1	no, access is allowed
data read	0	—	—	yes, no r permission
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 210*, 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.

21.3.2 Putting it all together and AHB error terminations

For each AHB slave port being monitored, the MPU performs a reduction-AND of all the individual (hit_b | error) terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, then a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see *Section 21.5: Application information*.

In event of a protection error, the MPU requires two distinct actions:

1. Intercepting the error during the AHB 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 AHB transaction to other portions of the platform.

If, instead, the access is allowed, then the MPU simply passes all “original” AHB signals to the slave device. In this case, from a functionality point of view, the MPU is fully transparent.

21.4 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.

21.5 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 21.2.2.4.4: 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 AHB bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EARn and MPU_EDRn registers. The error-terminated AHB bus cycle typically initiates some type of error response in the originating bus master. For example, 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].

22 Semaphores

22.1 Introduction

In a dual processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The semaphores module provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

22.1.1 Block diagram

Figure 211 is a simplified block diagram of the semaphores module that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

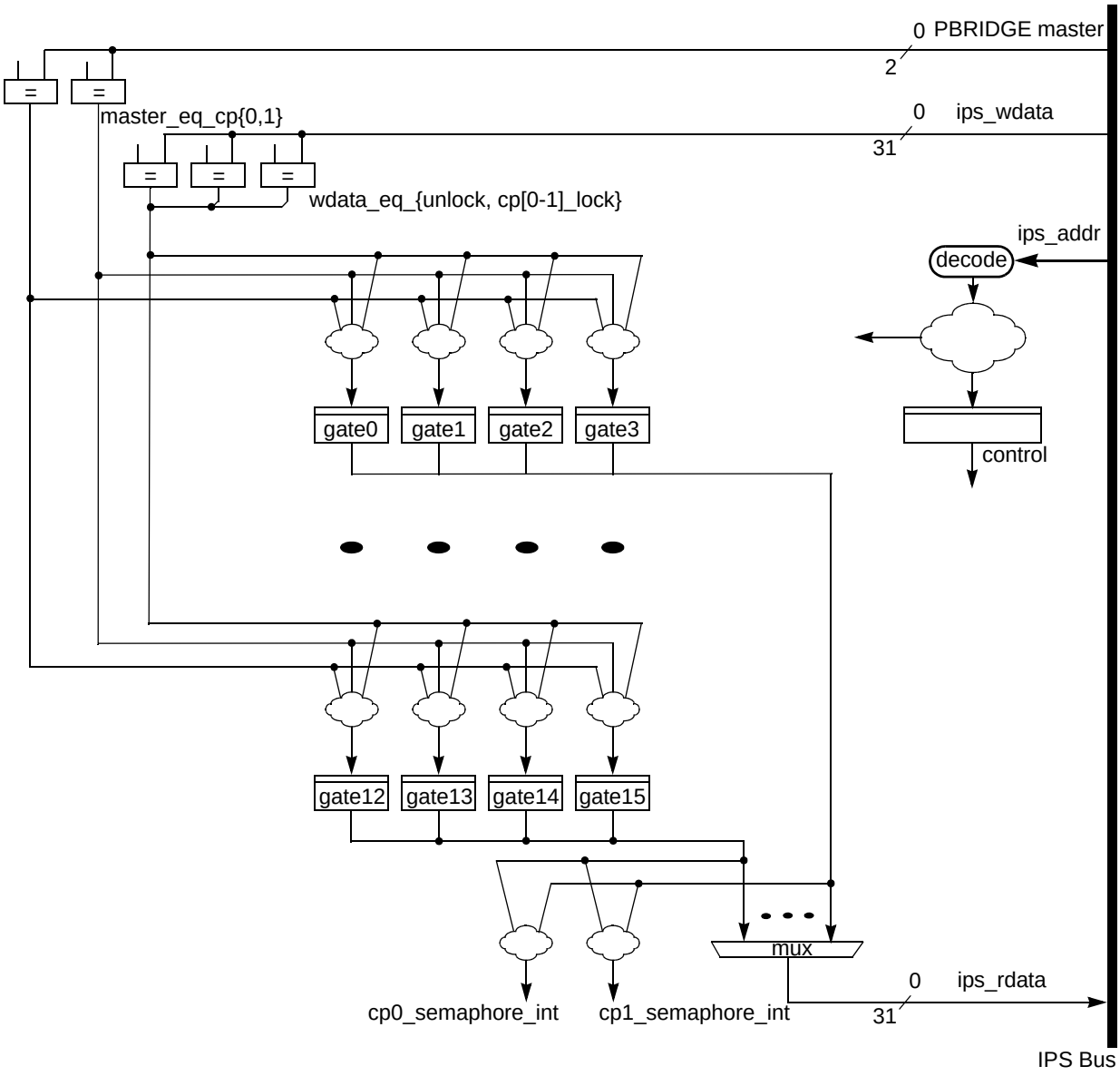


Figure 211. Semaphores block diagram

22.1.2 Features

The semaphores module implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration
  - Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes
- Three-state implementation
  - if gate = 0b00, then state = unlocked
  - if gate = 0b01, then state = locked by e200z4d (master ID = 0)
  - if gate = 0b10, then state = locked by e200z0h (master ID = 1)
  - a) Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
  - b) After it is locked, the gate must be unlocked by a write of zeroes from the locking processor
- Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked
- Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability

22.1.3 Modes of operation

The semaphores module does not support any special modes of operation.

22.2 Signal description

The semaphores module does not include any external signals.

22.3 Memory map and registers

This section provides a detailed description of all semaphores registers.

22.3.1 Module memory map

The semaphores programming model map is shown in *Table 201*. The address of each register is given as an offset to the semaphore base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 201. Semaphores memory map

Base address: 0xFFF2_4000		
Address offset	**Register**	**Location**
0x0000	SEMA4_Gate00—Semaphores gate 0	*on page 522*
0x0001	SEMA4_Gate01—Semaphores gate 1	*on page 522*
0x0002	SEMA4_Gate02—Semaphores gate 2	*on page 522*

Table 201. Semaphores memory map (continued)

Base address: 0xFFF2_4000		
Address offset	**Register**	**Location**
0x0003	SEMA4_Gate03—Semaphores gate 3	*on page 522*
0x0004	SEMA4_Gate04—Semaphores gate 4	*on page 522*
0x0005	SEMA4_Gate05—Semaphores gate 5	*on page 522*
0x0006	SEMA4_Gate06—Semaphores gate 6	*on page 522*
0x0007	SEMA4_Gate07—Semaphores gate 7	*on page 522*
0x0008	SEMA4_Gate08—Semaphores gate 8	*on page 522*
0x0009	SEMA4_Gate09—Semaphores gate 9	*on page 522*
0x000A	SEMA4_Gate10—Semaphores gate 10	*on page 522*
0x000B	SEMA4_Gate11—Semaphores gate 11	*on page 522*
0x000C	SEMA4_Gate12—Semaphores gate 12	*on page 522*
0x000D	SEMA4_Gate13—Semaphores gate 13	*on page 522*
0x000E	SEMA4_Gate14—Semaphores gate 14	*on page 522*
0x000F	SEMA4_Gate15—Semaphores gate 15	*on page 522*
0x0010–0x003F	Reserved	
0x040	SEMA4_CP0INE—Semaphores CP0 IRQ notification enable	*on page 522*
0x0042–0x0047	Reserved	
0x0048	SEMA4_CP1INE—Semaphores CP1 IRQ notification enable	*on page 522*
0x004A–0x07F	Reserved	
0x0080	SEMA4_CP0NTF—Semaphores CP0 IRQ notification	*on page 523*
0x008 2–00x087	Reserved	
0x0088	SEMA4_CP1NTF—Semaphores CP1 IRQ notification	*on page 522*
0x008A–0x00FF	Reserved	
0x0100	SEMA4_RSTGT—Semaphores reset gate	*on page 524*
0x0102—0x0103	Reserved	
0x0104	SEMA4_RSTNTF—Semaphores reset IRQ notification	*on page 525*
0x0106–0x3FFF	Reserved	

22.3.2 Register descriptions

This section lists the semaphores registers in address order and describes the registers and their bit fields.

22.3.2.1 Semaphores Gate *n* Register (SEMA4_GATE*n*)

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the write data operand must update the state of a single gate only. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Offset: SEMA4_BASE + *n* (*n* = 0, 1, 2,..., 15) Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	GTFSM	
W								
Reset	0	0	0	0	0	0	0	0

Figure 212. SEMA4 Gate n Register (SEMA4_GATEn)

Table 202. SEMA4_GATE*n* Field Descriptions

Field	Description
GTFSM	Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as: 00 The gate is unlocked (free). 01 The gate has been locked by the processor e200z4d. 10 The gate has been locked by the processor e200z0h. 11 This state encoding is never used and therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine. **Note:** The state of the gate reflects the last processor that locked it, which can be useful during system debug.

22.3.2.2 Semaphores Processor *n* IRQ Notification Enable (SEMA4_CP{0,1}INE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly-enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP*n*INE) and the interrupt request register (SEMA4_CP*n*NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP*n*INE registers, unimplemented bits read as zeroes and writes are ignored.

Offset: SEMA4_BASE + 0x0040 (SEMA4_CP0INE) Access: User read/write
SEMA4_BASE + 0x0048 (SEMA4_CP1INE)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INE0	INE1	INE2	INE3	INE4	INE5	INE6	INE7	INE8	INE9	INE10	INE11	INE12	INE13	INE14	INE15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 213. Semaphores Processor *n* IRQ Notification Enable (SEMA4_CP{0,1}INE)

Table 203. SEMA4_CP{0,1}NTF Field Descriptions

Field	Description
INE*n*	Interrupt Request Notification Enable *n*. This field is a bitmap to enable the generation of an interrupt notification from a failed attempt to lock gate *n*. 0 The generation of the notification interrupt is disabled. 1 The generation of the notification interrupt is enabled.

22.3.2.3 Semaphores Processor *n* IRQ Notification (SEMA4_CP{0,1}NTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

- When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
- After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
- When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN*n*] and SEMA4_CP1NTF[GN*n*].

Offset: SEMA4_BASE + 0x0080 (SEMA4_CP0NTF) Access: User read-only
SEMA4_BASE + 0x0088 (SEMA4_CP1NTF)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 214. Semaphores Processor *n* IRQ Notification (SEMA4_CP{0,1}NTF)

Table 204. SEMA4_CP{0,1}NTF Field Descriptions

Field	Description
GN*n*	Gate *n* Notification. This read-only field is a bitmap of the interrupt request notification from a failed attempt to lock gate *n*. 0 No notification interrupt generated. 1 Notification interrupt generated.

22.3.2.4 Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the semaphores module implements a secure reset mechanism which allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xE2; the least significant byte is a “don’t care” for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) which implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]) and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0100 (SEMA4_RSTGT) Access: User read/write

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

Figure 215. Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Table 205. SEMA4_RSTGT Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RSTGSM</td><td>Reset Gate Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as:
00 Idle, waiting for the first data pattern write.
01 Waiting for the second data pattern write.
10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state.
11 This state encoding is never used and therefore reserved.
Reads of the SEMA4_RSTGT register return the encoded state machine value. Note the RSTGSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.</td></tr>
<tr><td>RSTGMS</td><td>Reset Gate Bus Master. This 3-bit read-only field records the logical number of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.
<table><tr><th>Master</th><th>Master ID</th></tr><tr><td>e200z4d</td><td>0</td></tr><tr><td>e200z0h</td><td>1</td></tr></table></td></tr>
<tr><td>RSTGTN</td><td>Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write.
If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.</td></tr>
<tr><td>RSTGDP</td><td>Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xe2 while the second write requires RSTGDP = 0x1d.</td></tr>
</table>

22.3.2.5 Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, it is recognized that system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the semaphores module implements a secure reset mechanism which allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure reset mechanism requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a “don’t care” for this reference.
2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN])

specifies the notification(s) to be reset. This field can specify a single notification be cleared or that all notifications are cleared.

3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 (SEMA4_RSTNTF) Access: User read/write

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

Figure 216. Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

Table 206. SEMA4_RSTGT Field Descriptions

Field	Description
RSTNSM	Reset Notification Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The two-write sequence has completed. Generate the specified notification reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. 11 This state encoding is never used and therefore reserved. Reads of the SEMA4_RSTNTF register return the encoded state machine value. Note the RSTNSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.
RSTNMS	Reset Notification Bus Master. This 3-bit read-only field records the logical number of the bus master performing the notification reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs. **Master** / **Master ID** e200z4d / 0 e200z0h / 1
RSTNTN	Reset Notification Number. This 8-bit field specifies the specific IRQ notification state machine to be reset. This field is updated by the second write. If RSTNTN < 64, then reset the single IRQ notification machine defined by RSTNTN, else reset all the notifications.
RSTNDP	Reset Notification Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the notification reset mechanism. For the first write, RSTNDP = 0x47 while the second write requires RSTNDP = 0xb8.

22.4 Functional description

Multi-processor systems require a function that can be used to safely and easily provide a locking mechanism that is then used by system software to control access to shared data structures, shared hardware resources, and etc. These gating mechanisms are used by the software to serialize (and synchronize) writes to shared data and/or resources to prevent race conditions and preserve memory coherency between processes and processors.

For example, if processor X enters a section of code where shared data values are to be updated or read coherently, it must first acquire a semaphore. This locks, or closes, a software gate. After the gate has been locked, a properly architected software system does not allow other processes (or processors) to execute the same code segment or modify the shared data structure protected by the gate, that is, other processes/processors are locked out. Many software implementations include a spin-wait loop within the lock function until the locking of the gate is accomplished. After the lock has been obtained, processor X continues execution and updates the data values protected by the particular lock. After the updates are complete, processor X unlocks (or opens) the software gate, allowing other processes/processors access to the updated data values.

There are three important rules that must be followed for a correctly implemented system solution:

- All writes to shared data values or shared hardware resources must be protected by a gate variable.
- After a processor locks a gate, accesses to the shared data or resources by other processes/processors must be blocked. This is enforced by software conventions.
- The processor that locks a particular gate is the only processor that can unlock, or open, that gate.

Information in the hardware gate identifying the locking processor can be useful for system-level debugging.

The Hennessy/Patterson text on computer architecture offers this description of software gating:

> “One of the major requirements of a shared-memory architecture multiprocessor is being able to coordinate processes that are working on a common task. Typically, a programmer will use *lock variables* to synchronize the processes.
>
> The difficulty for the architect of a multiprocessor is to provide a mechanism to decide which processor gets the lock and to provide the operation that locks a variable. Arbitration is easy for shared-bus multiprocessors, since the bus is the only path to memory. The processor that gets the bus locks out all the other processors from memory. If the CPU and bus provide an atomic swap operation, programmers can create locks with the proper semantics. The adjective *atomic* is key, for it means that a processor can both read a location **and** set it to the locked value in the same bus operation, preventing any other processor from reading or writing memory.” [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 471-472]

The classic text continues with a description of the steps required to lock/unlock a variable using an atomic swap instruction.

> “Assume that 0 means unlocked and 1 means locked. A processor first reads the lock variable to test its state. A processor keeps reading and testing until the value indicates that the lock is unlocked. The processor then races against all other processes that were similarly “spin waiting” to see who can lock the variable first. All processes use a swap instruction that reads the old value and stores a 1 into the lock variable. The single winner will see the 0, and the losers will see a 1 that was placed there by the

winner. (The losers will continue to set the variable to the locked value, but that doesn't matter.) The winning processor executes the code after the lock and then stores a 0 into the lock when it exits, starting the race all over again. Testing the old value and then setting to a new value is why the atomic swap instruction is called *test and set* in some instruction sets." [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 472-473]

The sole drawback to a hardware-based semaphore module is the limited number of semaphores versus the infinite number that can be supported with Power Architecture reservation instructions.

22.4.1 Semaphore usage

Example 1: Inter-processor communication done with software interrupts and semaphores...

- The e200z0h uses software interrupts to tell the e200z4d that new data is available, or the e200z4d does the same to tell the e200z0h that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIU to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM not being updated, then the semaphore must first be locked, then the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
  - For example, if both cores want to update the same location, then the following sequence may occur.

    1. The e200z0h locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to the e200z4d.

    2. Before the e200z4d takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.

    3. The e200z4d software interrupt ISR reads the data sent to the e200z0h, not the data sent from the e200z0h, and performs an incorrect operation.
  - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores...

- The e200z4d wants to coherently read a section of shared memory.
- The e200z4d should check that the semaphore for the shared memory is not currently set.
- The e200z4d should set the semaphore for the shared memory to prevent the e200z0h from updating the shared memory.
- The e200z4d reads the required data, then unlock the semaphore.

22.5 Initialization information

The reset state of the semaphores module allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the module to immediately begin operation.

22.6 Application information

In an operational multi-core system, most interactions involving the Semaphores module involves reads and writes to the SEMA4_GATE*n* registers for implementation of the hardware-enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate
  - The processor performs a byte write of logical_processor_number + 1 to gate[i]
  - The processor reads back gate[i] and checks for a value of logical_processor_number + 1

    If the compare indicates the expected value

    then the gate is locked; proceed with the protected code segment

    else

    lock operation failed;

    repeat process beginning with byte write to gate[i] in spin-wait loop, or

    proceed with another execution path and wait for failed lock interrupt notification

A simple C-language example of a gatelock function is shown in *Example 1: Sample Gatelock Function*. This function follows the Hennessy/Patterson example.

Example 1 Sample Gatelock Function

```
#define UNLOCK      0
#define CP0_LOCK    1
#define CP2_LOCK    2

void gateLock (n)
int   n;    /* gate number to lock */
{
  int   i;
  int   current_value;
  int   locked_value;

  i = processor_number();  /* obtain logical CPU number */

  if (i == 0)
    locked_value = CP0_LOCK;
  else
    locked_value = CP1_LOCK;
```

```
  /* read the current value of the gate and wait until the state == UNLOCK
*/
  do {
    current_value = gate[n];
  } while (current_value != UNLOCK);

  /* the current value of the gate == UNLOCK. attempt to lock the gate for
this
     processor. spin-wait in this loop until gate ownership is obtained */
  do {
    gate[n] = locked_value;   /* write gate with processor_number + 1 */
    current_value = gate[n];  /* read gate to verify ownership was obtained
*/
  } while (current_value != locked_value);
}
```

- To unlock (open) a gate
  - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate

In this example, a reference to `processor_number()` is used to retrieve this hardware configuration value. Typically, the logical processor numbers are defined by a hardwired input vector to the individual cores. The exact method for accessing the logical processor number varies by architecture. For Power Architecture cores, there is a processor ID register (PIR) which is SPR 286 and contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: mfspr rx,286 where rx is the destination GPRn. Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., **`rdcpn rx`** for a read CPU number instruction.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CP*n*INE, SEMA4_ CP*n*NTF) are required. There is no required negation of the failed lock write notification interrupt as the request is automatically negated by the Semaphores module once the gate has been successfully locked by the failing processor.

Finally, in the event a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable(s) (SEMA4_CP*n*INE) bits be disabled before initiating the secure reset 2-write sequence to avoid any race conditions involving spurious notification interrupt requests.

22.7 DMA requests

There are no DMA requests associated with the IPS_Semaphore block.

22.8 Interrupt requests

The semaphore interrupt requests are connected to the interrupt controller as described in *Chapter 19: Interrupt Controller (INTC)*.

23 Performance Optimization

23.1 Introduction

The SPC564Bxx and SPC56ECxx contains several features that can influence the overall level of performance provided by the chip.

Some of these features may be initialized upon negation of reset by the Boot Assist Module (BAM), by a hardware state machine or by appropriate default register settings. Although the chip exits the reset state into a functional state it does not necessarily have the default optimum performance settings for any given application.

This chapter provides guidance for users to fully optimize their application to achieve the highest possible performance from the SPC564Bxx and SPC56ECxx. It provides a description of the areas that should be focused on when optimizing an application for performance by describing the features and recommending settings to be applied. It focuses on hardware configurations although certain aspects of the application software such as compiler settings and optimizations will be discussed.

23.2 Features

The SPC564Bxx and SPC56ECxx has the following hardware features that can be configured to impact the overall performance of the chip:

- Branch Prediction
  - Branch Target Buffer
  - Branch Prediction Control
- Frequency-Modulated Phase-Locked Loop (FMPLL)
- Platform Flash Memory Controller (PFC)
  - Flash access wait state and address pipelining control
  - Flash instruction prefetching
  - Flash data prefetching
- Crossbar switch
- System cache
  - Instruction cache on e200z4d
- Memory Management Unit

Further application level features can impact the application performance:

- Hardware Single Precision Floating point
- Signal Processing Extension (SPE-APU)
- Variable Length Encoding (VLE)
- Compiler optimizations

Further factors that impact the overall application performance are the use of the intelligent peripherals:

- Use of DMA rather than CPU to transfer data efficiently
- Use of DMA service requests rather than CPU interrupts to avoid software polling
- Off-loading tasks from the CPU to the eDMA
- Careful allocation of cache usage for code ranges.

Different items in this list will have different performance impacts in a real system. Features like the crossbar switch, system cache, the FMPLL and the flash access times tend to provide the most significant performance impacts in terms of hardware settings.

The subsequent sections in this chapter describe how to configure and use these features.

23.3 Configuring hardware features

23.3.1 Branch target buffer (BTB)

23.3.1.1 Description

To resolve branch instructions and improve the accuracy of branch predictions both the e200z4d and e200z0h cores implement a dynamic branch prediction mechanism using a branch target buffer (BTB), a fully associative address cache of branch target addresses. Its purpose is to accelerate the execution of software loops with some potential change of flow within the loop body. In addition, the BTB on the e200z4d has a subroutine call stack that speeds up indirect branches.

23.3.1.2 Recommended configuration

By default, the BTB is disabled following reset on both the e200z4d and the e200z0h. It is controlled by the Branch Unit Control and Status Register (BUCSR). The BTB's contents should be flushed and invalidated by writing BUCSR[BBFI] = 1, and it may be enabled by subsequently writing BUCSR[BPEN] = 1.

Additional control is available by configuring e200z0h HID0[BPRED] or e200z4d BUCSR[BPRED] and BUCSR[BALLOC] to control whether forward or backward branches (or both) are candidates for entry into the BTB, and thus for branch prediction. By default the HID0[BPRED], BUCSR[BPRED] and BUCSR[BALLOC] fields are set to 0b00, which enables forward and backward branch prediction. It is recommended to not disable branch prediction although for extremely fine tuning of a given application the optimum setting of HID0[BPRED], BUCSR[BPRED] and BUCSR[BALLOC] should be assessed.

0	BBFI	0	BALLOC	0	BPRED	BPEN
0 - 21	22	23 - 24	25 - 26	27 - 28	29 - 30	31

e200z4d SPR - 1013; Read/Write; Reset - 0x0

Figure 217. Branch Unit Control and Status Register (BUCSR)

Table 207. BUCSR field descriptions

Field	Description
BBFI	Branch target buffer flash invalidate When set, BBFI flash clears the valid bit of all BTB entries; clearing occurs regardless of the value of the enable bit (BPEN). **Note**: BBFI is always read as 0.
BALLOC	Branch Target Buffer Allocation Control 00: Branch Target Buffer allocation for all branches is enabled. 01: Branch Target Buffer allocation is disabled for backward branches. 10: Branch Target Buffer allocation is disabled for forward branches. 11: Branch Target Buffer allocation is disabled for both branch directions. This field controls BTB allocation for branch acceleration when BPEN = 1. Note that BTB hits are not affected by the settings of this field. Note that for branches with AA = '1', the MSB of the displacement field is still used to indicate forward/backward, even though the branch is absolute.
BPRED	Branch Prediction Control (Static) 00: Branch predicted taken on BTB miss for all branches. 01: Branch predicted taken on BTB miss only for forward branches. 10: Branch predicted taken on BTB miss only for backward branches. 11: Branch predicted not taken on BTB miss for both branch directions. This field controls operation of static prediction mechanism on a BTB miss. Unless disabled, fetching of the predicted target location will be performed for branch acceleration. BPRED operates independently of BPEN, and with a BPEN setting of 0, will be used to perform static prediction of all unresolved branches. Note that BTB hits are not affected by the settings of this field. Note that for certain applications, setting BPRED to a non-default value may result in improved performance.
BPEN	Branch target buffer (BTB) enable 0: BTB prediction disabled. No hits are generated from the BTB and no new entries are allocated. Entries are not automatically invalidated when BPEN is cleared; BBFI controls entry invalidation. 1: BTB prediction enabled (enables BTB to predict branches).

23.3.2 Frequency-modulated PLL

23.3.2.1 Description

The frequency-modulated phase-locked loop (FMPLL) allows the user to generate high speed system clocks from a crystal oscillator or external clock generator. Further, the FMPLL supports programmable frequency modulation of the system clock. This module is typically configured early in the initialization code to ensure satisfactory performance levels are achieved.

23.3.2.2 Recommended configuration

After reset the SPC564Bxx and SPC56ECxx chip uses the Fast Internal RC Oscillator (FIRC) as its system clock (approximately 16 MHz). Typically, the source of the system clock is changed to the FMPLL to provide acceptable performance. *Section 6.7: Frequency-modulated phase-locked loop (FMPLL)*, provides details on how the FMPLL should be initialized in an application. The maximum frequency of operation for this chip is specified in the chip data sheet.

System performance cannot be linearly extrapolated with system frequency, as is often the expectation. It is due to the insertion of additional flash wait states as system frequency increases that system performance does not scale linearly. Take care to ensure that the

correct internal and/or external flash configuration is chosen for the selected system frequency. The specific flash access times to be applied are detailed in *Section 35.4: Platform Flash Controller*.

23.3.3 Flash memory bus interface unit

23.3.3.1 Description

The Platform Flash Memory Controller (PFC) interfaces the system bus to the flash memory array controller. The PFC contains prefetch buffers and a prefetch controller which, if enabled, speculatively prefetches sequential lines of data from the flash array into the buffer. Prefetch buffer hits allow zero-wait state responses.

The Platform Flash Configuration Registers (PFCRx) control access to the internal flash array. Its settings define the number of cycles required to access the array, access times, and how the prefetch buffering scheme operates.

Following negation of reset and execution of the BAM, the instruction and data prefetching is disabled, and the number of cycles required to access the internal flash array is set to its maximum value of fifteen additional wait states.

23.3.3.2 Recommended configuration

As the operating frequency of the chip is set by configuring the FMPLL (see *Section 23.3.2: Frequency-modulated PLL*), the number of cycles required to access the internal array should be configured accordingly. Note that the flash PFCRx registers cannot be altered by code executing from the flash array. Code for configuring the flash should be executed from a separate memory array i.e copied to and executed from system RAM.

Section 35.4: Platform Flash Controller, documents the register fields used to configure flash wait state settings. The “Platform flash controller electrical characteristics” section of the chip data sheet contains the specific values for the flash wait state settings for a given operating frequency. This also provides recommendations for the prefetch buffer settings. Note that the PFCRx settings may vary between revisions of the SPC564Bxx and SPC56ECxx.

23.3.4 Crossbar switch

23.3.4.1 Description

The multi-port crossbar switch (XBAR) supports simultaneous connections between master ports and slave ports. The XBAR allows for concurrent transactions to occur from any master port to any slave port. 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. By default, requesting masters are granted access based on a fixed priority. A round-robin priority mode also is available.

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. In order to maximize data throughput it is essential to keep arbitration delays to a minimum. The configuration of the crossbar can have implications for the performance of a system and particular care should be taken when assigning master priorities in a fixed priority application. Further, by correctly parking saves on relevant masters the initial access times to the slaves can be minimized by negating any initial arbitration penalties.

23.3.4.2 Recommended configuration

The specific settings for a given situation are application dependent and thus should be assessed by the user. The primary flash port is fixed to the e200z4d instruction bus master to maximise execution speed for that core however the assignment of the second flash port will depend on which other masters are accessing the flash the most. Best performance may be obtained by prioritising the instruction bus of the e200z0h, the eDMA or the data bus of the e200z4d. Similarly assignment of the two system RAM ports depend on how the cores and eDMA use the RAMs.

More details of the XBAR register configuration can be found in *Section 20.3: XBAR registers*.

23.3.5 Cache

23.3.5.1 Description

The SPC564Bxx and SPC56ECxx e200z4d provides an 8 KB instruction, 2-way or 4-way set-associative, Harvard cache design with a 32-byte line size. The cache is disabled by default after reset.

The cache improves system performance by providing low-latency instructions to the e200z4d instruction pipelines, which decouples processor performance from system memory performance. There are several stages to enabling the cache. Not only does the cache itself have to be invalidated then enabled, but memory regions upon which it can operate must be configured in the MMU to permit cache access.

23.3.5.2 Recommended configuration

The exact usage of cache is application dependent but some general guidelines for using cache to improve performance in a typical application are listed below:

- Enable instruction cache for all internal and external memories that code is being executed from.
- Consider locking critical performance routines in cache.

The process of enabling the instruction cache involves first invalidating the cache (by setting L1CSR1[ICINV]) then when invalidation is completed (L1CSR1[ICINV, ICABT] = 0) enabling the cache (by setting L1CSR1[ICE]).

The L1CSR1 special purpose register is detailed below.

0	ICECE	ICEI	0	ICEDT	0	ICUL	ICLO	ICLFC	ICLOA	ICEA	0	ICABT	ICINV	ICE
0 - 14	15	16	17	18 - 19	20	21	22	23	24	25 - 26	27 - 28	29	30	31

SPR - 1011; Read/Write; Reset - 0x0

Figure 218. L1 Cache Control and Status Register 1 (L1CSR1)

Table 208. L1CSR1 field descriptions

Field	Description
ICECE	Instruction Cache Error Checking Enable
ICEI	Instruction Cache Error Injection Enable

Table 208. L1CSR1 field descriptions (continued)

Field	Description
ICEDT	Instruction Cache Error Detection Type
ICUL	Instruction Cache Unable to Lock
ICLO	Instruction Cache Lock Overflow
ICLFC	Instruction Cache Lock Bits Flash Clear
ICLOA	Instruction Cache Lock Overflow Allocate
ICEA	Instruction Cache Error Action
ICABT	Instruction Cache Operation Aborted Indicates a Cache Invalidate or a Cache Lock Bits Flash Clear operation was aborted prior to completion. This bit is set by hardware on an aborted condition, and will remain set until cleared by software writing 0 to this bit location.
ICINV	Instruction Cache Invalidate 0: No cache invalidate 1: Cache invalidation operation When written to a ‘1’, a cache invalidation operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while an invalidation operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while an invalidation operation is in progress will be ignored. Cache invalidation operations require approximately 36 cycles to complete. Invalidation occurs regardless of the enable (ICE) value. During cache invalidations, the parity check bits are written with a value dependent on the ICEDT selection. ICEDT should be written with the desired value for subsequent cache operation when ICINV is set to ‘1’ for proper operation of the cache.
ICE	Instruction Cache Enable 0: Cache is disabled 1: Cache is enabled When disabled, cache lookups are not performed for instruction accesses. Other L1CSR0 cache control operations are still available.

Note that configuration of the cache has to be performed in conjunction with configuration of the Memory Management Unit. See section *Section 23.3.6: e200z4d Memory Management Unit (MMU)*.

23.3.6 e200z4d Memory Management Unit (MMU)

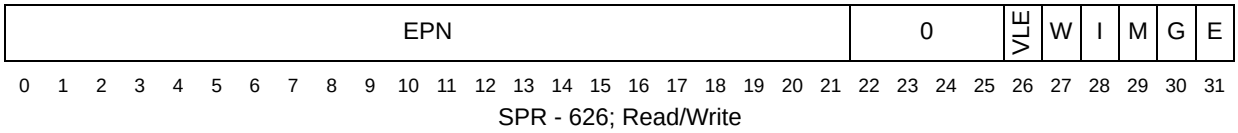
23.3.6.1 Description

The e200z4d Memory Management Unit is a 32-bit Power Architecture compliant implementation which provides functionality that includes address translation and application of access attributes to memory ranges defined in Translation Lookaside Buffer entries. Although the MMU does not directly impact performance, it is within the MMU that memory regions are configured to permit the use of system cache to improve performance and Variable Length Encoding (VLE) to enhance code density. Thus it is essential that the MMU is correctly configured to ensure optimal application performance is achieved.

23.3.6.1.1 Recommended configuration

The core uses MMU Assist Registers (MASx) which are special purpose registers to facilitate reading, writing and searching the Translation Lookaside Buffer (TLB) entries. These MAS registers are software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions.

There are several MMU Assist Register registers (MAS0–3) that require configuring. Specifically, the MAS2 register contains the fields to control whether a specified memory region described by the valid TLB Entry is cache inhibited or whether VLE encoding is valid.



SPR - 626; Read/Write

Figure 219. MMU Assist Register 2 (MAS2)

Table 209. MAS2 field descriptions

Field	Description
EPN	Effective page number [0:21]
VLE	VLE 0: This page is a standard BookE page 1: This page is a VLE page
W	Write-through required
I	Cache inhibited 0: This page is considered cacheable 1: This page is considered cache-inhibited
M	Memory coherence required
G	Memory coherence required
E	Endianness

23.4 Application software

23.4.1 Compiler optimizations

The most significant opportunity for influencing the performance of a given application is by compiler and linker optimizations. Optimizing is a trade off between code size and performance. Typically higher performance of the application comes at the expense of larger code size. Compilers use a host of features, such as loop unrolling, function inlining and application profile feedback to make the desired trade-offs between enhanced performance and minimized code size.

The data in *Figure 220* shows the effects of compiler optimization on a simple application. In this case, the Dhrystone benchmark was run under three conditions:

- Optimized for small code size
- Optimized for high performance
- A trade-off between code size and performance

Although this is an extreme example, it highlights how significant the role of the compiler and linker is in determining the overall performance of an application.

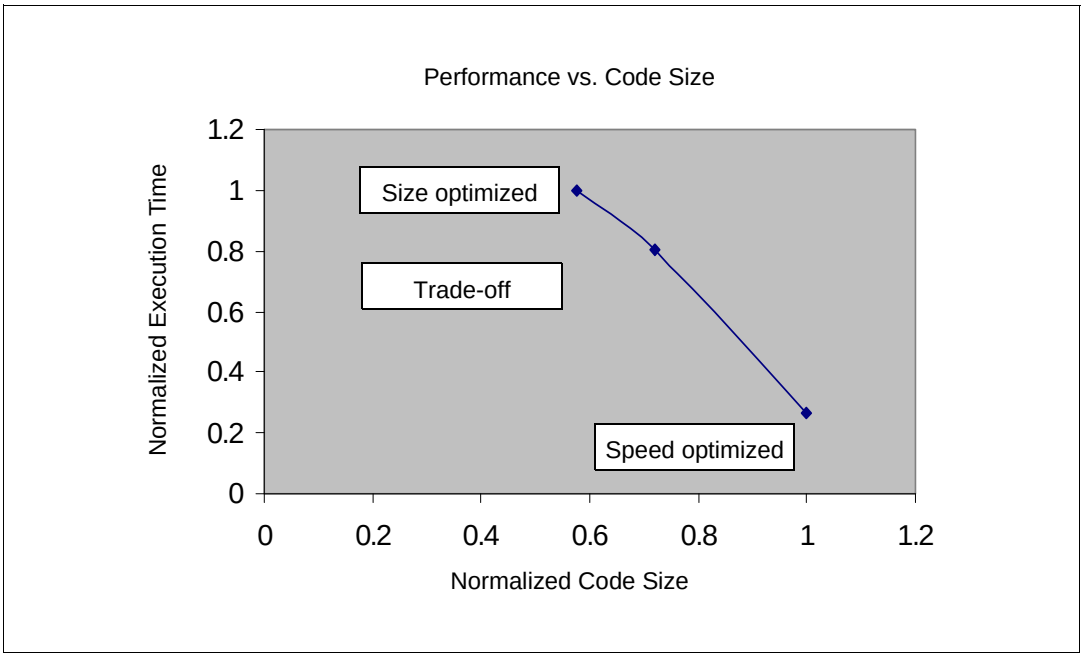


Figure 220. Influence of compiler settings on application performance and code size

Note: Data measured using Dhrystone version 2.1 run on a Power Architecture based powertrain chip using a standard commercial compiler.

The compiler optimizations do not necessarily have to be applied to the entire application. Analysis of an application can identify time critical functions that may subsequently be targeted for performance optimization, without incurring the impact of optimizing the entire application.

There are several other aspects of the compiler and linker that should be considered. In particular, the use of Small Data Areas (SDAs, sometimes referred to as Special Data Areas) can make a significant performance improvement. Refer to compiler documentation for usage guidelines on Small Data Areas.

23.4.2 e200z4d Signal Processing Extension

To further optimize time critical functions, the Signal Processing Extension Auxiliary Processing Unit (SPE-APU) may be used on the e200z4d. The SPE-APU provides a set of Single Instruction Multiple Data (SIMD) instructions. These SIMD instructions typically involve performing the same operation on multiple data elements stored within a single 64-bit register. Through the implementation of SIMD instructions, including vector multiply and accumulate (MAC) instructions, the SPE APU provides Digital Signal Processing (DSP) functionality. This can be used to accelerate signal processing routines, such as Finite Impulse Response (FIR), Infinite Impulse Response (IIR) and Discrete Fourier Transforms (DFT). A more general benefit of the SPE instruction set is the ability to load/store 64-bits of data in single instruction. Thus highly load/store intensive functions make good candidates for SPE optimization.

0	UCLE	SPE	0	WE	CE	0	EE	PR	FP	ME	0	DE	0	IS	DS	0	RI	0
0:4	5	6	7:12	13	14	15	16	17	18	19	20:21	22	23:25	26	27	28:29	30	31

Read / Write; Reset - 0x0

Figure 221. Machine State Register (MSR)

Table 210. MSR field descriptions

Field	Description
UCLE	User Cache Lock Enable
SPE	SPE Available 0: Execution of SPE APU vector instructions is disabled; SPE Unavailable exception taken instead, and SPE bit is set in ESR. 1: Execution of SPE APU vector instructions is enabled.
WE	Wait State (Power management) enable
CE	Critical Interrupt Enable
EE	External Interrupt Enable
PR	Problem State
FP	Floating-Point Available
ME	Machine Check Enable
DE	Debug Interrupt Enable
IS	Instruction Address Space
DS	Data Address Space
RI	Recoverable Interrupt

23.4.3 Hardware single precision floating point

The SPE-APU also supports 32-bit IEEE®-754 single-precision floating-point formats, and supports vector and scalar single-precision floating-point operations. Most compiler vendors include libraries that can emulate floating point functionality. However, by specifying the correct compiler options, the single precision floating point instructions may be used.

To enable use of hardware floating point the MSR[SPE] field must be set. See *Section 23.4.2: e200z4d Signal Processing Extension*, for register details.

23.4.4 Variable Length Encoding (VLE)

In addition to the base Power Architecture instruction set support, the e200z4d core also implements the VLE APU, providing improved code density. This setting is permanently enabled on the e200z0h. The VLE APU can be viewed as a supplement to the existing Power Architecture instruction set that can be conditionally applied to a portion of, or an entire application for which improved code density is desired.

Using it is straightforward:

1. Select the appropriate compiler target and option to generate VLE code.
2. Configure the Memory Management Unit (MMU) to specify VLE attributes for the relevant MMU pages. Refer to the register description in *Section 23.3.6: e200z4d Memory Management Unit (MMU)*.

VLE-enabled cores run both Power Architecture and VLE instruction encodings on a page by page basis, with pages defined by the MMU. The reduction is code size is typically between 25% and 30%.

23.5 Peripherals and general application guidelines

Optimizing the chip configuration and compiler setup is only one part of optimizing an entire application. Correct use of the peripherals can also dramatically improve overall system performance. In particular, use of the interrupt controller, the Enhanced Direct Memory Access (eDMA), the Cross Triggering Unit (CTU) and the e200z0h can off-load significant work from the e200z4d.

For example, eDMA may be used to shift data to avoid unnecessary CPU loading. Most peripheral modules can generate eDMA requests to trigger data transfers. An example of a typical application is to use the eDMA to move CAN messages from one FlexCAN module to another. If pre-processing is required before this move the 200z0h core may be a better choice. For ADC triggering and sample result handling the CTU can be used to avoid CPU intervention.

Section 23.6: Performance optimization checklist, provides several system level examples of how to optimize an application.

23.6 Performance optimization checklist

23.6.1 Hardware configuration

Table 211. Performance optimization — hardware configuration

Description	Register(s)	Details
Branch Target Buffer	Flush with BUCSR[BBFI] Enable with BUCSR[BPEN]	Flush and enable to improve accuracy of branch predictions.
Branch Prediction	BUCSR[BPRED]/HID0[BPRED] BUCSR[BALLOC]	Consider fine tuning of BTB operation for specific applications.
System Frequency	FMPLL_CR FMPLL_MR	Select desired frequency and frequency modulation taking into account the performance impact of additional wait states.
Flash Wait States	FPCR0[APC, WW, RWSC]	Refer to Flash chapter for FPCR settings for FMPLL frequency ranges.
Flash Prefetching	FPCR[DPFEN, IPFEN, PFLIM, BFEN]	Enable prefetching for instructions. Prefetching for data should be assessed for the specific application.
Flash Prefetch Algorithm	FPCR[BCFG]	Allocate buffers to data and/or instructions. Fine tune for specific applications.

Table 211. Performance optimization — hardware configuration (continued)

Description	Register(s)	Details
Crossbar Switch	SGPCRn and MPRn	Configure ports according to most likely master to access a slave. In single core designs assign the e200z4d data port to the second flash port. In multi-core designs prioritise the flash and RAM ports to the heaviest users
Cache	Invalidate Icache with L1CSR1[ICINV] Enable Icache with L1CSR1[ICE]	Invalidate and the enable the cache for instructions.
Memory Management Unit	TLB_MAS2[VLE, I]	Configure relevant pages for cache and VLE by setting MMU TLB attributes.

23.6.2 Software configuration

Table 212. Performance optimization — software configuration

Description	Registers	Details
Compiler optimization	—	Use the features of the compiler to select the optimum trade off between code size and performance improvements.
Hardware Single Precision Floating Point	Enable with MSR[SPE]	Set compiler switches to specify using hardware single precision floating point as opposed to software emulation.
Signal Processing Extensions	Enable with MSR[SPE]	Take advantage of the SPE-APU to encode time critical functions using SPE assembly code.
Variable Length Encoding	Enabled with TLB_MAS2[VLE]	Set compiler switches and configure the MMU to take advantage of the VLE-APU.

23.6.3 Peripherals and general application guidelines

- Use eDMA rather than the core to move data where possible. Most peripherals can generate eDMA requests to shift data.
  - Use DMA to fill and empty communication chips buffers.
- Shift loading from the CPU to the e200z0h and CTU whenever possible.
  - The e200z0h can handle interrupts and scheduling for comunications peripherals.
  - The CTU can trigger the ADC directly with no need for CPU interruption.
- Avoid software polling and allow peripherals to trigger interrupts or request eDMA servicing.
  - Use hardware instead of software vectored interrupts to reduce latency.
  - Trigger eDMA requests rather than interrupting the CPU to move data/results.

24 System Integration Unit Lite (SIUL)

24.1 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.

24.2 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 222* provides a block diagram of the SIUL and its interfaces to other system components.

The module provides dedicated general-purpose pads that can be configured as either inputs or outputs.

- When 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.

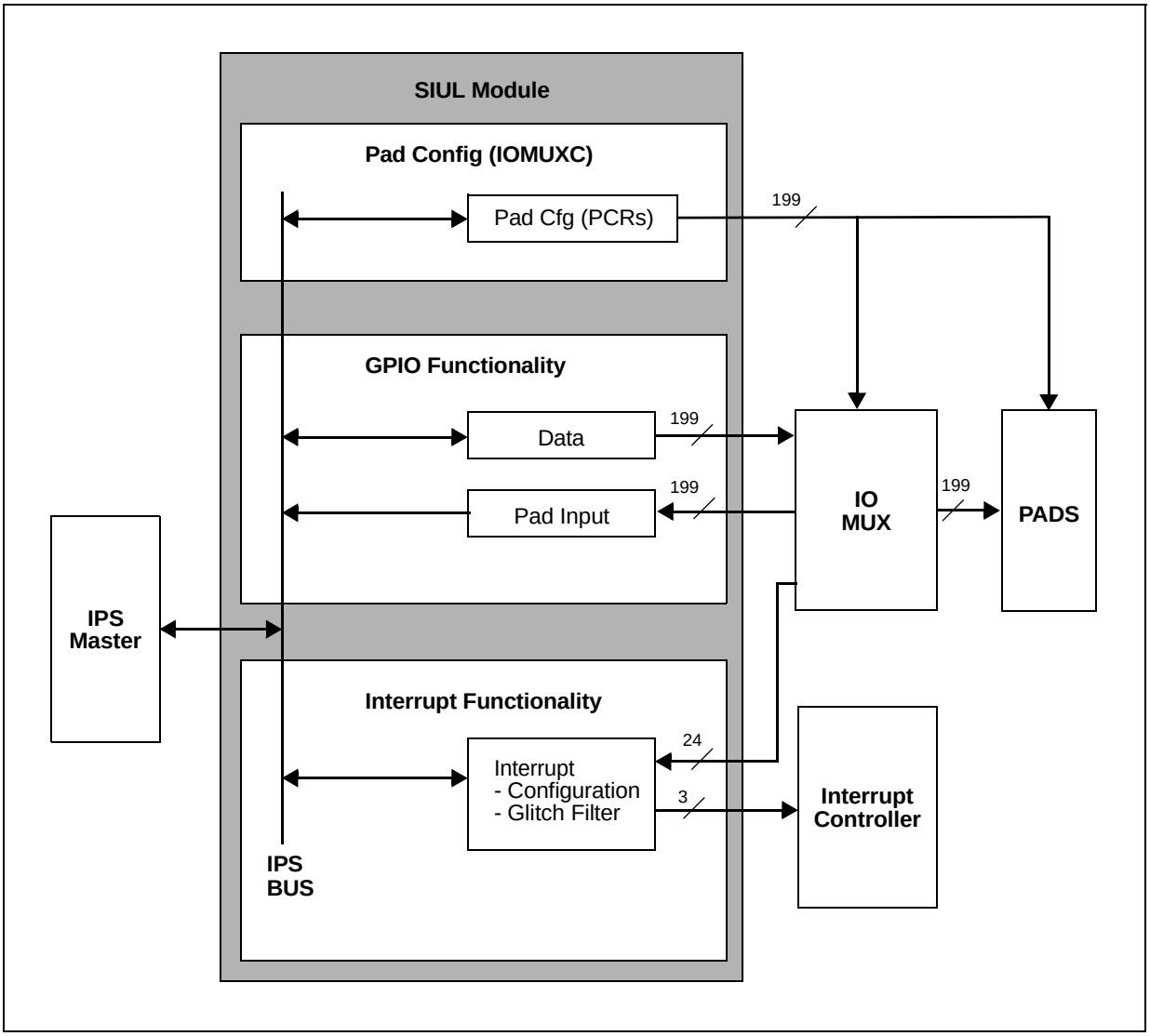


Figure 222. System Integration Unit Lite block diagram

24.3 Features

The System Integration Unit Lite supports these distinctive features:

- GPIO
  - GPIO function on up to 199 I/O pins
  - Dedicated input and output registers for most GPIO pins[h]
- External interrupts
  - 3 interrupt vectors dedicated to 24 external interrupts
  - 24 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control
  - Pad Selection for multiplexed inputs registers

24.4 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.

Table 213 lists the external pins used by the SIUL.

Table 213. SIUL signal properties

GPIO[0:198] category	Name	I/O direction	Function
System configuration	GPIO[0:198]	I/O	General-purpose input/output
External interrupt	PA[3], PA[6:8], PA[11:12], PA[14], PC[2:5], PC[12], PC[14:15], PE[2], PE[4], PE[6:7], PE[10], PE[12], PE[14], PF[15], PG[1],PG[8]	Input	Pins with External Interrupt Request functionality. Please refer to the signal description chapter of this reference manual for details.

24.4.1 Detailed signal descriptions

24.4.1.1 General-purpose I/O pins (GPIO[0:198])

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.

h. Some device pins, e.g., analog pins, do not have both input and output functionality.

24.4.1.2 External interrupt request input pins (EIRQ[0:23])

The EIRQ[0:23] 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.

24.5 Memory map and register description

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

24.5.1 SIUL memory map

Table 214 gives an overview of the SIUL registers implemented.

Table 214. SIUL memory map

Base address: 0xC3F9_0000		
Address offset	**Register**	**Location**
0x0004	MCU ID Register #1 (MIDR1)	*on page 547*
0x0008	MCU ID Register #2 (MIDR2)	*on page 548*
0x000C–0x0013	Reserved	
0x0014	Interrupt Status Flag Register (ISR)	*on page 549*
0x0018	Interrupt Request Enable Register (IRER)	*on page 549*
0x001C–0x0027	Reserved	
0x0028	Interrupt Rising-Edge Event Enable Register (IREER)	*on page 550*
0x002C	Interrupt Falling-Edge Event Enable Register (IFEER)	*on page 550*
0x0030	Interrupt Filter Enable Register (IFER)	*on page 551*
0x0034–0x003F	Reserved	
0x0040–0x01CC	Pad Configuration Registers (PCR0–PCR198)	*on page 552*
0x01CE–0x04FF	Reserved	
0x500–0x543	Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI64_67)	*on page 554*
0x0544–0x05FF	Reserved	
0x0600–0x06C7	GPIO Pad Data Output Registers (GPDO0_3–GPDO196_199)	*on page 560*
0x06C8–0x07FF	Reserved	
0x0800–0x08C7	GPIO Pad Data Input Registers (GPDI0_3–GPDI196_199)	*on page 560*
0x08C8–0x0BFF	Reserved	
0x0C00–0x0C18	Parallel GPIO Pad Data Out Registers (PGPDO0–PGPDO6)	*on page 561*
0x0C1C–0x0C3F	Reserved	
0x0C40–0x0C58	Parallel GPIO Pad Data In Register (PGPDI0–PGPDI6)	*on page 560*
0x0C5C–0x0C7F	Reserved	

Table 214. SIUL memory map (continued)

Base address: 0xC3F9_0000		
Address offset	**Register**	**Location**
0x0C80–0x0CB0	Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO12)	*on page 562*
0x0CB4–0x0FFF	Reserved	
0x1000–0x105C	Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)	*on page 564*
0x1060–0x107F	Reserved	
0x1080	Interrupt Filter Clock Prescaler Register (IFCPR)	*on page 565*
0x1084–0x10FF	Reserved	
0x1100–0x113F	Parallel Input Select Register (PISR0—PISR15)	*on page 565*
0x1140–0x11FF	Reserved	
0x1200	DSPI Input Select Register (DISR)	*on page 575*
0x1204–0x3FF	Reserved	

Note: A transfer error will be issued when trying to access completely reserved register space.

24.5.2 Register protection

Individual registers in System Integration Unit Lite can be protected from accidental writes using the Register Protection module (*Chapter 36: Register Protection*). The following registers can be protected:

- *Section 24.5.3.4: Interrupt Request Enable Register (IRER)*
- *Section 24.5.3.5: Interrupt Rising-Edge Event Enable Register (IREER)*
- *Section 24.5.3.6: Interrupt Falling-Edge Event Enable Register (IFEER)*
- *Section 24.5.3.7: Interrupt Filter Enable Register (IFER)*, entire PortA, PortB[0:3] and PortC[2:15
- Interrupt Filter Enable Register (IFER)
- *Section 24.5.3.8: Pad Configuration Registers (PCR0–PCR198)*
- *Section 24.5.3.9: Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI64_67)*
- *Section 24.5.3.15: Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)*
- *Section 24.5.3.16: Interrupt Filter Clock Prescaler Register (IFCPR)*

24.5.3 Register description

24.5.3.1 MCU ID Register #1 (MIDR1)

This register holds identification information about the device.

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	Reset values depend on the device and package type as shown in *Table 215*															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	PKG					0	0	MAJOR_MASK				MINOR_MASK			
W																
Reset	Reset values depend on the device and package type as shown in *Table 215*															

Figure 223. MCU ID Register #1 (MIDR1)

Table 215. MIDR1 field descriptions

Field	Description
PARTNUM[15:0]	MCU Part Number Device part number of the MCU. 0101_0110_0100_0100: 1.5MB 0101_0110_0100_0101: 2MB 0101_0110_0100_0110: 3MB For the full part number this field needs to be combined with MIDR2[PARTNUM[23:16]].
PKG	Package Settings Can be read by software to determine the package type that is used for the particular device: 0b10000: 208 BGA 0b01100: 256 BGA 0b10001: 176-pin LQFP 0b10101: 208-pin LQFP
MAJOR_MASK	Major Mask Revision Counter starting at 0x0. Incremented each time when there is a resynthesis.
MINOR_MASK	Minor Mask Revision Counter starting at 0x0. Incremented each time a mask change is done.

24.5.3.2 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	Reset values depend on the device and package type.														

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PARTNUM[23:16]								0	0	0	EE(1)	0	0	0	FR(1)
W																
Reset	Reset values depend on the device and package type.															

Figure 224. MCU ID Register #2 (MIDR2)

1. The reset values of both EE and FR bit is always same.

Table 216. MIDR2 field descriptions

Field	Description
SF	Manufacturer 0: Reserved 1: STMicroelectronics
FLASH_SIZE_1	Coarse granularity for flash memory size 4 bits used to define major Flash memory size Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2 0b0011: 128 KB 0b0100: 256 KB 0b0101: 512 KB 0b0110: 1 MB 0b0111: 2 MB
FLASH_SIZE_2	Fine granularity for flash memory size Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2 0b0000: 0 x (FLASH_SIZE_1 / 8) 0b0010: 2 x (FLASH_SIZE_1 / 8) 0b0100: 4 x (FLASH_SIZE_1 / 8)
PARTNUM[23:16]	ASCII character in MCU Part Number 0x42h: Character ‘B’ 0x43h: Character ‘C’ 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
FR	FlexRay Present 0: No FlexRay is present 1: FlexRay is present

24.5.3.3 Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

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	EIF[23:16]							
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF[15:0]															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 225. Interrupt Status Flag Register (ISR)

Table 217. 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

24.5.3.4 Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging to the interrupt controller.

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	EIRE[23:16]							
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	EIRE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 226. Interrupt Request Enable Register (IRER)

Table 218. IRER field descriptions

Field	Description
EIRE[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.

24.5.3.5 Interrupt Rising-Edge Event Enable Register (IREER)

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

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	IREE[23:16]							
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	IREE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 227. Interrupt Rising-Edge Event Enable Register (IREER)

Table 219. 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

24.5.3.6 Interrupt Falling-Edge Event Enable Register (IFEER)

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

Offset:0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									IFEE[23:16]							
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	IFEE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 228. Interrupt Falling-Edge Event Enable Register (IFEER)

Table 220. 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.

24.5.3.7 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.

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	IFE[23:16]							
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	IFE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 229. Interrupt Filter Enable Register (IFER)

Table 221. 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 232* for details on how the filter works.

24.5.3.8 Pad Configuration Registers (PCR0–PCR198)

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.

Offsets:0x0040–0x01CC (199 registers) 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[2:0]			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)

Figure 230. Pad Configuration Registers (PCRx)

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, *Table 223* describes the PCR depending on the pad type (pad types are defined in *Chapter 4: Signal Description*). The bits in shaded fields are not implemented for the particular I/O type. The PA field selecting the number of alternate functions may or may not be present depending on the number of alternate functions actually mapped on the pad.

Table 222. 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 (Pad with GPIO and digital alternate function)		SMC	APC		PA[1:0]		OBE	IBE			ODE				WPE	WPS
M, F (Pad with GPIO and digital alternate function)		SMC	APC		PA[1:0]		OBE	IBE			ODE			SRC	WPE	WPS
Pad with GPIO and more than 4 alternate function		SMC	APC	PA[2:0]			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 223. 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.
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	Pad Output Assignment This field is used to select the function that is allowed to drive the output of a multiplexed pad. 000: Alternative Mode 0 — GPIO 001: Alternative Mode 1 — See the signal description chapter 010: Alternative Mode 2 — See the signal description chapter 011: Alternative Mode 3 — See the signal description chapter 100: Alternative Mode 4 — See the signal description chapter **Note:** Number of bits depends on the actual number of actual alternate functions. Please refer to datasheet.
OBE	Output Buffer Enable This bit enables the output buffer of the pad if 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 1: Pad is configured as medium or fast (depending on the pad) 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

24.5.3.9 Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI64_67)

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.

Note: This register can be addressed as either a single 32-bit field or it can be addressed as four individual 8-bit fields.

Offsets: 0x500–0x540 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

Figure 231. Pad Selection for Multiplexed Inputs Register (PSMI0_3)

Table 224. PSMI field descriptions

Field	Description
PADSEL0–3, PADSEL4–7, ... PADSEL64–67	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function. See *Table 225*.

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 225. Peripheral input pin selection

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI0_3	PADSEL0	0x500	CAN1RX / FlexCAN 1	000: PCR[35] 001: PCR[43] 010: PCR[95] 011: PCR[157] 100: PCR[159] 101: PCR[0]
	PADSEL1	0x501	CAN2RX / FlexCAN 2	000: PCR[73] 001: PCR[89] 010: PCR[165]
	PADSEL2	0x502	CAN3RX / FlexCAN 3	000: PCR[36] 001: PCR[73] 010: PCR[89] 011: PCR[167] 100: PCR[173] 101: PCR[1]
	PADSEL3	0x503	CAN4RX / FlexCAN 4	000: PCR[35] 001: PCR[43] 010: PCR[95] 011: PCR[157] 100: PCR[161]
PSMI4_7	PADSEL4	0x504	CAN5RX / FlexCAN 5	000: PCR[64] 001: PCR[97] 010: PCR[163]
	PADSEL5	0x505	SCK_0 / DSPI 0	001: PCR[14] 011: PCR[15]
	PADSEL6	0x506	CS0_0 / DSPI 0	000: PCR[14] 001: PCR[15] 010: PCR[27]
	PADSEL7	0x507	SCK_1 / DSPI 1	000: PCR[34] 001: PCR[68] 010: PCR[114] 011: PCR[173]

Table 225. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI8_11	PADSEL8	0x508	SIN_1 / DSPI 1	000: PCR[36] 001: PCR[66] 010: PCR[112] 011: PCR[175] 100: PCR[10]
	PADSEL9	0x509	CS0_1 / DSPI 1	000: PCR[35] 001: PCR[61] 010: PCR[69] 011: PCR[115] 100: PCR[4] 101: PCR[174]
	PADSEL10	0x50A	SCK_2 / DSPI 2	000: PCR[46] 001: PCR[78] 010: PCR[105]
	PADSEL11	0x50B	SIN_2 / DSPI 2	000: PCR[44] 001: PCR[76]
PSMI12_15	PADSEL12	0x50C	CS0_2 / DSPI 2	000: PCR[47] 001: PCR[79] 010: PCR[82] 011: PCR[104] 100: PCR[140]
	PADSEL13	0x50D	E0UC[3] / eMIOS 0	000: PCR[3] 001: PCR[27] 010: PCR[40]
	PADSEL14	0x50E	E0UC[4] / eMIOS 0	000: PCR[4] 001: PCR[28]
	PADSEL15	0x50F	E0UC[5] / eMIOS 0	000: PCR[5] 001: PCR[29]
PSMI16_19	PADSEL16	0x510	E0UC[6] / eMIOS 0	000: PCR[6] 001: PCR[30]
	PADSEL17	0x511	E0UC[7] / eMIOS 0	000: PCR[7] 001: PCR[31] 010: PCR[41]
	PADSEL18	0x512	E0UC[10] / eMIOS 0	000: PCR[10] 001: PCR[80]
	PADSEL19	0x513	E0UC[11] / eMIOS 0	000: PCR[11] 001: PCR[81]

Table 225. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI20_23	PADSEL20	0x514	E0UC[12] / eMIOS 0	000: PCR[44] 001: PCR[82]
	PADSEL21	0x515	E0UC[13] / eMIOS 0	000: PCR[45] 001: PCR[83] 010: PCR[0]
	PADSEL22	0x516	E0UC[14] / eMIOS 0	000: PCR[46] 001: PCR[84] 010: PCR[8]
	PADSEL23	0x517	E0UC[22] / eMIOS 0	000: PCR[70] 001: PCR[72] 010: PCR[85]
PSMI24_27	PADSEL24	0x518	E0UC[23] / eMIOS 0	000: PCR[71] 001: PCR[73] 010: PCR[86]
	PADSEL25	0x519	E0UC[24] / eMIOS 0	000: PCR[60] 001: PCR[106] 010: PCR[75]
	PADSEL26	0x51A	E0UC[25] / eMIOS 0	000: PCR[61] 001: PCR[107]
	PADSEL27	0x51B	E0UC[26] / eMIOS 0	000: PCR[62] 001: PCR[108]
PSMI28_31	PADSEL28	0x51C	E0UC[27] / eMIOS 0	000: PCR[63] 001: PCR[109]
	PADSEL29	0x51D	SCL / I2C	000: PCR[11] 001: PCR[19]
	PADSEL30	0x51E	SDA / I2C	000: PCR[10] 001: PCR[18]
	PADSEL31	0x51F	LIN3RX / LINFlexD_3	000: PCR[8] 001: PCR[75] 010:PCR[167]
PSMI32_35	PADSEL32	0x520	SCK_3 / DSPI 3	000: PCR[100] 001: PCR[124]
	PADSEL33	0x521	SIN_3 / DSPI 3	000: PCR[101] 001: PCR[139]
	PADSEL34	0x522	CS0_3 / DSPI 3	000: PCR[99] 001: PCR[125] 010: PCR[140]
	PADSEL35	0x523	SCK_4 / DSPI 4	000: PCR[109] 001: PCR[126] 010: PCR[133] 011:PCR[171]

Table 225. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI36_39	PADSEL36	0x524	SIN_4 / DSPI 4 Reserved	000: PCR[106] 001: PCR[142] 010: PCR[169]
	PADSEL37	0x525	CS0_4 / DSPI 4 Reserved	000: PCR[107] 001: PCR[123] 010: PCR[134] 011: PCR[143] 100: PCR[172]
	PADSEL38	0x526	E0UC[0] / eMIOS 0	000: PCR[0] 001: PCR[14]
	PADSEL39	0x527	E0UC[1] / eMIOS 0	000: PCR[1] 001: PCR[15]
PSMI40_43	PADSEL40	0x528	E0UC[28] / eMIOS 0	000: PCR[12] 001: PCR[128]
	PADSEL41	0x529	E0UC[29] / eMIOS 0	000: PCR[13] 001: PCR[129]
	PADSEL42	0x52A	E0UC[30] / eMIOS 0	000: PCR[16] 001: PCR[18] 010: PCR[130]
	PADSEL43	0x52B	E0UC[31] / eMIOS0	000: PCR[17] 001: PCR[19] 010: PCR[131]
PSMI44_47	PADSEL44	0x52C	E1UC[1] / eMIOS 1	000: PCR[111] 001: PCR[89] 010: PCR[164]
	PADSEL45	0x52D	E1UC[2] / eMIOS 1	000: PCR[112] 001: PCR[90]
	PADSEL46	0x52E	E1UC[3] / eMIOS 1	000: PCR[113] 001: PCR[91]
	PADSEL47	0x52F	E1UC[4] / eMIOS 1	000: PCR[114] 001: PCR[95]
PSMI48_51	PADSEL48	0x530	E1UC[5] / eMIOS 1	000: PCR[115] 001: PCR[123]
	PADSEL49	0x531	E1UC[17] / eMIOS 1	000: PCR[104] 001: PCR[127]
	PADSEL50	0x532	E1UC[18] / eMIOS 1	000: PCR[105] 001: PCR[148]
	PADSEL51	0x533	E1UC[25] / eMIOS 1	000: PCR[92] 001: PCR[124]

Table 225. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping[1]
PSMI52_55	PADSEL52	0x534	E1UC[26] / eMIOS 1	000: PCR[93] 001: PCR[125]
	PADSEL53	0x535	E1UC[27] / eMIOS 1	000: PCR[94] 001: PCR[126]
	PADSEL54	0x536	E1UC[28] / eMIOS 1	000: PCR[38] 001: PCR[132]
	PADSEL55	0x537	E1UC[29] / eMIOS 1	000: PCR[39] 001: PCR[133]
PSMI56_59	PADSEL56	0x538	E1UC[30] / eMIOS 1Reserved	000: PCR[74] 001: PCR[103] 010: PCR[134]
	PADSEL57	0x539	E1UC[31] / eMIOS 1Reserved	000: PCR[36] 001: PCR[106] 010: PCR[135]
	PADSEL58	0x53A	LIN2RX / LINFlexD_2	000: PCR[41] 001: PCR[11] 010:PCR[165]
	PADSEL59	0x53B	LIN4RX / LINFlexD_4 Reserved	000: PCR[6] 001: PCR[91]
PSMI60_63	PADSEL60	0x53C	LIN5RX / LINFlexD_5	000: PCR[4] 001: PCR[93]
	PADSEL61	0x53D	LIN8RX / LINFlexD_8	000: PCR[111] 001: PCR[129] 010:PCR[163]
	PADSEL62	0x53E	LIN0RX / LINFlexD_0	000: PCR[19] 001: PCR[17]
	PADSEL63	0x53F	E1UC[0] / eMIOS 1	000: PCR[110] 001: PCR[163]
PSMI64_67	PADSEL64	0x540	CS0_5 / DSPI 5	000: PCR[134] 001: PCR[146]
	PADSEL65	0x541	CS0_6 / DSPI 6	000: PCR[107] 001: PCR[134] 010:PCR[146]
	PADSEL66	0x542	SIN_7 / DSPI 7	000: PCR[117] 001: PCR[175]
	PADSEL67	0x543	CS0_7 / DSPI 7	000: PCR[119] 001: PCR[146]

1. See the signal description chapter of this reference manual for correspondence between PCR and pinout

24.5.3.10 GPIO Pad Data Output Registers (GPDO0_3–GPDO196_199)

These registers are used to set or clear GPIO pads. Each pad data out bit can be controlled separately with a byte access.

Offsets:0x0600–0x06C4 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

Figure 232. Port GPIO Pad Data Output Register 0–3 (GPDO0_3)

Table 226. 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

24.5.3.11 GPIO Pad Data Input Registers (GPDI0_3–GPDI196_199)

These registers are used to read the GPIO pad data with a byte access.

Offsets:0x0800–0x08C4 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

Figure 233. Port GPIO Pad Data Input Register 0–3 (GPDI0_3)

Table 227. GPDO0_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

24.5.3.12 Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO6)

SPC564Bxx and SPC56ECxx 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 refer to datasheet.

Table 228 shows the locations and structure of the PGPDO*x* registers.

Table 228. PGPDO0 – PGPDO6 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														
0x0C08	PGPDO2	Port E																Port F														
0x0C0C	PGPDO3	Port G																Port H														
0x0C10	PGPDO4	Port I																Port J														
0x0C14	PGPDO5	Port K																Port L														
0x0C18	PGPDO6	Port M																Reserved														

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 228*, 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].

24.5.3.13 Parallel GPIO Pad Data In Register (PGPDI0 – PGPDI6)

The SIU_PGPDI registers are similar in operation to the PGPDIO registers, described in the previous section (*Section 24.5.3.12: Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO6)*) 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 229 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 229. PGPDI0 – PGPDI6 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														
0x0C50	PGPDI4	Port I																Port J														
0x0C54	PGPDI5	Port K																Port L														
0x0C58	PGPDI6	Port M																Reserved														

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 229*, 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].

24.5.3.14 Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO12)

The MPGPDO*x* registers are similar in operation to the PGPDO*x* ports described in *Section 24.5.3.12: Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO6)*, 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 230 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 230. MPGPDO0 – MPGPDO12 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)														
0x0CA0	MPGPDO8	MASK8 (Port I)																MPPDO8 (Port I)														
0x0CA4	MPGPDO9	MASK9 (Port J)																MPPDO9 (Port J)														
0x0CA8	MPGPDO10	MASK10 (Port K)																MPPDO10 (Port K)														
0x0CAC	MPGPDO11	MASK11 (Port L)																MPPDO11 (Port)														
0x0CB0	MPGPDO12	MASK12 (Port M)								Reserved								MPPDO12 (Port M)								Reserved						

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 230, 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 231. MPGPDO0..MPGPDO12 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 *Section 24.5.3.10: GPIO Pad Data Output Registers (GPDO0_3–GPDO196_199)*. 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]

24.5.3.15 Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)

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.

Offsets: 0x1000–0x105C 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

Figure 234. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)

Table 232. IFMC field descriptions

Field	Description
MAXCNTx	Maximum Interrupt Filter Counter setting Filter Period = T(CK)*3 (for 2 < MAXCNT < 6) Filter Period = T(CK)*MAXCNTx (for MAXCNT = 6,7,.... 15) For MAXCNT = 0, 1, 2 the filter behaves as ALL PASS filter. MAXCNTx can be 0 to 15; T(CK): Prescaled Filter Clock Period, which is IRC clock prescaled to IFCP value; T(IRC): Basic Filter Clock Period: 62.5 ns (F = 16 MHz).

24.5.3.16 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.

Offset: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

Figure 235. Interrupt Filter Clock Prescaler Register (IFCPR)

Table 233. 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.

24.5.3.17 Parallel Input Select Register (PISR0—PISR15)

The SIUL includes 4 groups of 16:1 multiplexers (32 multiplexers in each group) that are used to route 32 eMIOS PWM output signals to a 32-bit serialisation register in each of 4 DSPI modules:

- 32 eMIOS_0 signals serialized to DSPI_0
- 32 eMIOS_1 signals serialized to DSPI_1
- 32 eMIOS_0 signals serialized to DSPI_2
- 32 eMIOS_1 signals serialized to DSPI_3

For maximum flexibility, rather than fixing the 32 channels within an eMIOS module to a specified output, the source channel can be chosen from one of 16 options via the 16:1 multiplexer. This is organised such that the default channel is always the same as the output number (for example, eMIOS channel 0 for output 0) but you can also choose one of the previous 8 channel numbers or one of the next 7. The IPSx field determines the channel selection for each of the 32 channels and is based on a signed integer value (–8 to +7). The IPSx bitfields are contained within a set of 32-bit PISR (Parallel Input Select Registers) with 8 IPSx fields per register. This means that for each group, there are 4 PISR registers needed to contain the full 32 IPSx bitfields:

- PISR[0..3] for eMIOS_0 to DSPI_0
- PISR[4..7] for eMIOS_1 to DSPI_1
- PISR[8..11] for eMIOS_0 to DSPI_2
- PISR[12..15] for eMIOS_1 to DSPI_3

The PISRs are shown in *Figure 236* through *Figure 251*.

Offset: 0x1100 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS0				IPS1				IPS2				IPS3			
Reset	0	0	0	0	0	0	0	0	0	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	IPS4				IPS5				IPS6				IPS7			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 236. Parallel Input Select Register 0 (PISR0)

Offset: 0x1104 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS8				IPS9				IPS10				IPS11			
Reset	0	0	0	0	0	0	0	0	0	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	IPS12				IPS13				IPS14				IPS15			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 237. Parallel Input Select Register 1 (PISR1)

Offset: 0x1108 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS16				IPS17				IPS18				IPS19			
Reset	0	0	0	0	0	0	0	0	0	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	IPS20				IPS21				IPS22				IPS23			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 238. Parallel Input Select Register 2 (PISR2)

Offset: 0x110C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS24				IPS25				IPS26				IPS27			
Reset	0	0	0	0	0	0	0	0	0	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	IPS28				IPS29				IPS30				IPS31			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 239. Parallel Input Select Register 3 (PISR3)

Offset: 0x1110 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS0				IPS1				IPS2				IPS3			
Reset	0	0	0	0	0	0	0	0	0	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	IPS4				IPS5				IPS6				IPS7			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 240. Parallel Input Select Register 4 (PISR4)

Offset: 0x1114 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS8				IPS9				IPS10				IPS11			
Reset	0	0	0	0	0	0	0	0	0	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	IPS12				IPS13				IPS14				IPS15			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 241. Parallel Input Select Register 5 (PISR5)

Offset: 0x1118 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS16				IPS17				IPS18				IPS19			
Reset	0	0	0	0	0	0	0	0	0	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	IPS20				IPS21				IPS22				IPS23			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 242. Parallel Input Select Register 6 (PISR6)

Offset: 0x111C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS24				IPS25				IPS26				IPS27			
Reset	0	0	0	0	0	0	0	0	0	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	IPS28				IPS29				IPS30				IPS31			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 243. Parallel Input Select Register 7 (PISR7)

Offset: 0x1120 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS0				IPS1				IPS2				IPS3			
Reset	0	0	0	0	0	0	0	0	0	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	IPS4				IPS5				IPS6				IPS7			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 244. Parallel Input Select Register 8 (PISR8)

Offset: 0x1124 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS8				IPS9				IPS10				IPS11			
Reset	0	0	0	0	0	0	0	0	0	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	IPS12				IPS13				IPS14				IPS15			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 245. Parallel Input Select Register 9 (PISR9)

Offset: 0x1128 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS16				IPS17				IPS18				IPS19			
Reset	0	0	0	0	0	0	0	0	0	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	IPS20				IPS21				IPS22				IPS23			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 246. Parallel Input Select Register 10 (PISR10)

Offset: 0x112C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS24				IPS25				IPS26				IPS27			
Reset	0	0	0	0	0	0	0	0	0	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	IPS28				IPS29				IPS30				IPS31			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 247. Parallel Input Select Register 11 (PISR11)

Offset: 0x1130 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS0				IPS1				IPS2				IPS3			
Reset	0	0	0	0	0	0	0	0	0	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	IPS4				IPS5				IPS6				IPS7			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 248. Parallel Input Select Register 12 (PISR12)

Offset: 0x1134 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS8				IPS9				IPS10				IPS11			
Reset	0	0	0	0	0	0	0	0	0	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	IPS12				IPS13				IPS14				IPS15			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 249. Parallel Input Select Register 13 (PISR13)

Offset: 0x1138 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS16				IPS17				IPS18				IPS19			
Reset	0	0	0	0	0	0	0	0	0	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	IPS20				IPS21				IPS22				IPS23			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 250. Parallel Input Select Register 14 (PISR14)

Offset: 0x1140 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS24				IPS25				IPS26				IPS27			
Reset	0	0	0	0	0	0	0	0	0	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	IPS28				IPS29				IPS30				IPS31			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 251. Parallel Input Select Register 15 (PISR15)

Figure 252 shows the multiplexing scheme for the 1st group. This is mirrored for each subsequent group.

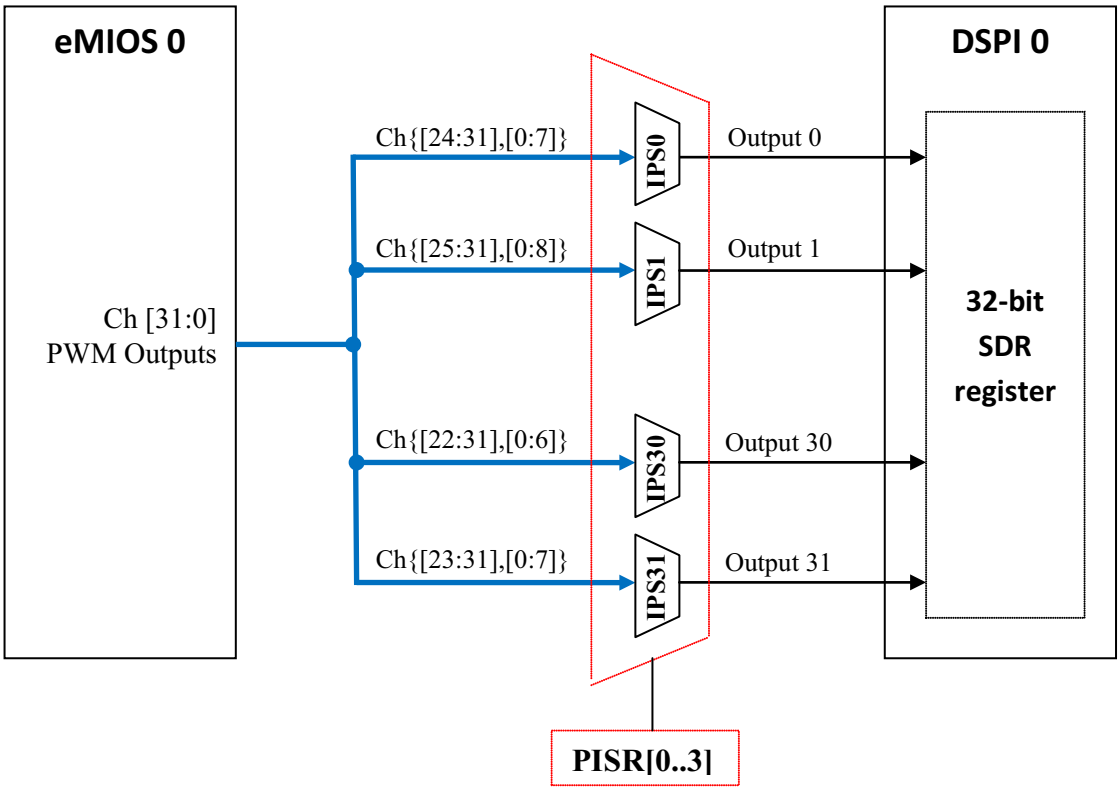


Figure 252. eMIOS_0 to DSPI_0 serialization

The table below shows the available eMIOS channel inputs for each output channel and provides an easy way of configuring the multiplexing. To use the table:

1. Select which group you are interested in (from one of the 4 PISR register set columns).
2. Select the output you wish to configure.
   Each row of the table is a corresponding output. The IPSx number is the same as the

output number. The number of the PISR register that needs to be configured can be extracted from the "PISR register set column".

3. Choose the input eMIOS channel for that output from the "Resultant eMIOS channel number column".
   The IPSx multiplexer value can be obtained for that output by tracing up to the top of the column.
4. Write the IPSx register value obtained in step 3 to the PISR and IPSx bitfield defined in step (2).
5. Repeat for all outputs.

Table 234. eMIOS channel inputs for each output channel

PISR number				IPSx mux field	IPSx field value and integer equivalent															
					8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
					-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
eMIOS_0 to DSPI_0	eMIOS_1 to DSPI_1	eMIOS_0 to DSPI_3	eMIOS_1 to DSPI_4		Resultant eMIOS channel number															
0	4	8	12	0	24	25	26	27	28	29	30	31	0	1	2	3	4	5	6	7
0	4	8	12	1	25	26	27	28	29	30	31	0	1	2	3	4	5	6	7	8
0	4	8	12	2	26	27	28	29	30	31	0	1	2	3	4	5	6	7	8	9
0	4	8	12	3	27	28	29	30	31	0	1	2	3	4	5	6	7	8	9	10
0	4	8	12	4	28	29	30	31	0	1	2	3	4	5	6	7	8	9	10	11
0	4	8	12	5	29	30	31	0	1	2	3	4	5	6	7	8	9	10	11	12
0	4	8	12	6	30	31	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	4	8	12	7	31	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	5	9	13	8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	5	9	13	9	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	5	9	13	10	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	5	9	13	11	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	5	9	13	12	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	5	9	13	13	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	5	9	13	14	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Table 234. eMIOS channel inputs for each output channel (continued)

1	5	9	13	15	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2	6	10	14	16	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	6	10	14	17	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
2	6	10	14	18	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	6	10	14	19	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
2	6	10	14	20	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
2	6	10	14	21	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
2	6	10	14	22	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
2	6	10	14	23	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
3	7	11	15	24	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
3	7	11	15	25	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	0
3	7	11	15	26	18	19	20	21	22	23	24	25	26	27	28	29	30	31	0	1
3	7	11	15	27	19	20	21	22	23	24	25	26	27	28	29	30	31	0	1	2
3	7	11	15	28	20	21	22	23	24	25	26	27	28	29	30	31	0	1	2	3
3	7	11	15	29	21	22	23	24	25	26	27	28	29	30	31	0	1	2	3	4
3	7	11	15	30	22	23	24	25	26	27	28	29	30	31	0	1	2	3	4	5
3	7	11	15	31	23	24	25	26	27	28	29	30	31	0	1	2	3	4	5	6

24.5.3.18 DSPI Input Select Register (DISR)

A device shall allow DSPIs 0, 1, 2 and 3 to be serial and parallel chained. The DISR registers specifies the source of each DSPI data input, slave select, clock input, and trigger input to enable serial and parallel chaining of the DSPI modules.

Offset: 0x1200 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SINSEL0		SSSSEL0		SCKSEL0		TRIGSEL0		SINSEL1		SSSSEL1		SCKSEL1		TRIGSEL1	
Reset	0	0	0	0	0	0	0	0	0	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	SINSEL2		SSSSEL2		SCKSEL2		TRIGSEL2		SINSEL3		SSSSEL3		SCKSEL3		TRIGSEL3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 253. DSPI Input Select Register (DISR)

Table 235. DISR field descriptions

Field	Description
SINSEL0	DSPI0 data input select. Specifies the source of DSPI0 data input. 00 DSPI0_SIN pin 01 DSPI1_SOUT 10 DSPI2_SOUT 11 DSPI3_SOUT
SSSEL0	DSPI0 slave select input select. Specifies the source of the DSPI0 slave select input. 00 DSPI0_PCS[0]/SS pin 01 DSPI1_PCS[0] (Master) 10 DSPI2_PCS[0] (Master) 11 DSPI3_PCS[0] (Master)
SCKSEL0	DSPI0 clock input select. Specifies the source of the DSPI0 clock input. 00 DSPI0 SCK pin 01 DSPI1_SCK (Master) 10 DSPI2_SCK (Master) 11 DSPI3_SCK (Master)
TRIGSEL0	DSPI0 trigger input select. Specifies the source of the DSPI_0 trigger input for master or slave mode. 01 DSPI1_PCS[4]/MTRIG 10 DSPI2_PCS[4]/MTRIG 11 DSPI3_PCS[4]/MTRIG
SINSEL1	DSPI1 data input select. Specifies the source of DSPI1 data input. 00 DSPI1_SIN pin 01 DSPI0_SOUT 10 DSPI2_SOUT 11 DSPI3_SOUT

Table 235. DISR field descriptions (continued)

Field	Description
SSSEL1	DSPI1 slave select input select. Specifies the source of the DSPI1 slave select input. 00 DSPI1_PCS1[0]/SS pin 01 DSPI0_PCS[0] (Master) 10 DSPI2_PCS[0] (Master) 11 DSPI3_PCS[0] (Master)
SCKSEL1	DSP1 clock input select. Specifies the source of the DSPI1 clock input. 00 DSPI1_SCK pin 01 DSPI0_SCK(Master) 10 DSPI2_SCK (Master) 11 DSPI3_SCK (Master)
TRIGSEL1	DSPI1 trigger input select. Specifies the source of the DSPI1 trigger input for master or slave mode. 01 DSPI0_PCS[4]/MTRIG 10 DSPI2_PCS[4]/MTRIG 11 DSPI3_PCS[4]/MTRIG
SINSEL2	DSPI2 data input select. Specifies the source of DSPI2 data input. 00 DSPI2_SIN pin 01 DSPI0_SOUT 10 DSPI1_SOUT 11 DSPI3_SOUT
SSSEL2	DSPI2 slave select input select. Specifies the source of the DSPI2 slave select input. 00 DSPI2_PCS2[0]/SS pin 01 DSPI0_PCS[0] (Master) 10 DSPI1_PCS[0] (Master) 11 DSPI3_PCS[0] (Master)
SCKSEL2	DSPI2 clock input select. Specifies the source of the DSPI2 clock input. 00 DSPI2_SCK pin 01 DSPI0_SCK (Master) 10 DSPI1_SCK (Master) 11 DSPI3_SCK (Master)
TRIGSEL2	DSPI2 trigger input select. Specifies the source of the DSPI2 trigger input for master or slave mode. 01 DSPI0_PCS[4]/MTRIG 10 DSPI1_PCS[4]/MTRIG 11 DSPI3_PCS[4]/MTRIG
SINSEL3	DSPI3 data input select. Specifies the source of DSPI3 data input. 00 DSPI3_SIN pin 01 DSPI0_SOUT 10 DSPI1_SOUT 11 DSPI2_SOUT
SSSEL3	DSPI3 slave select input select. Specifies the source of the DSPI3 slave select input. 00 DSPI3_PCS[0]/SS pin 01 DSPI0_PCS[0] (Master) 10 DSPI1_PCS[0] (Master) 11 DSPI2_PCS[0] (Master)

Table 235. DISR field descriptions (continued)

Field	Description
SCKSEL3	DSPI3 clock input select. Specifies the source of the DSPI3 clock input. 00 DSPI3_SCK pin 01 DSPI0_SCK (Master) 10 DSPI1_SCK (Master) 11 DSPI2_SCK (Master)
TRIGSEL3	DSPI3 trigger input select. Specifies the source of the DSPI3 trigger input for master or slave mode. 01 DSPI0_PCS[4]/MTRIG 10 DSPI1_PCS[4]/MTRIG 11 DSPI2_PCS[4]/MTRIG

24.6 Functional description

24.6.1 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 24.5.3.8: Pad Configuration Registers (PCR0–PCR198)*) 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

24.6.2 General purpose input and output pads (GPIO)

The SIUL manages up to 199 GPIO pads organized as ports that can be accessed for data reads and writes as 32, 16 or 8-bit.[i]

As shown in *Figure 254*, all port accesses are identical with each read or write being performed only at a different location to access a different port width.

i. There are exceptions. Some pads, e.g., precision analog pads, are input only.

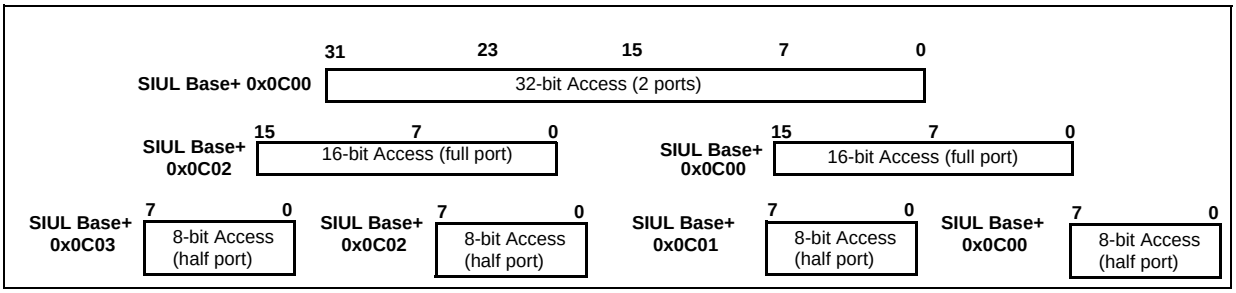


Figure 254. Data Port example arrangement showing configuration for different port width accesses

The SIUL has separate data input (GPDIn_n, see *Section 24.5.3.11: GPIO Pad Data Input Registers (GPDI0_3–GPDI196_199)*) and data output (GPDOn_n, see *Section 24.5.3.10: GPIO Pad Data Output Registers (GPDO0_3–GPDO196_199)*) 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.

The allocation of what input function is connected to the pin is defined by the PSMI registers (PCR*n*, see *Section 24.5.3.9: Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI64_67)*)

24.6.3 External interrupts

The SIUL supports 24 external interrupts, EIRQ0–EIRQ23. In the signal description chapter of this reference manual, mapping is shown for external interrupts to pads.

The SIUL supports three interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

Refer to *Figure 255* for an overview of the external interrupt implementation.

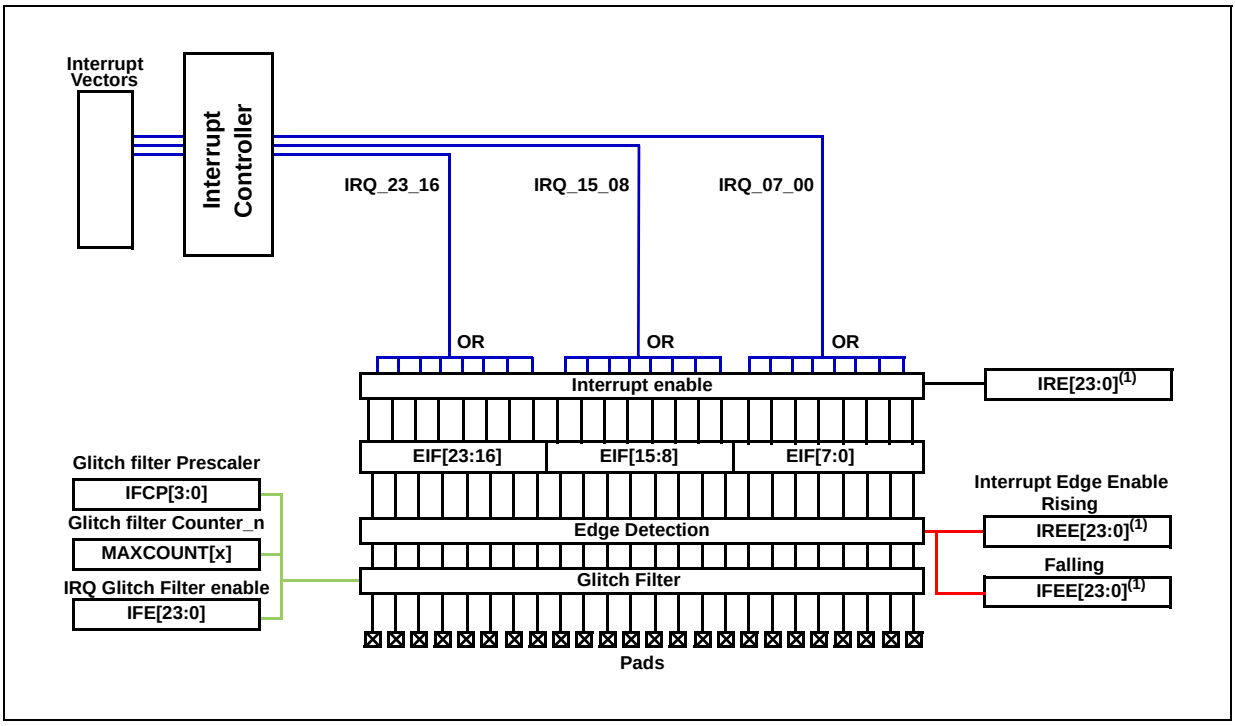


Figure 255. External interrupt pad diagram

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 ISR. This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.

24.7 Pin muxing

For pin muxing, please refer to the signal description chapter of this reference manual.

25 Inter-Integrated Circuit Bus Controller Module (I2C)

25.1 Introduction

25.1.1 Overview

The Inter-Integrated Circuit (I2C™ 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.

25.1.2 Features

The I2C module has the following key features:

- Compatible with I2C 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

25.1.3 Block diagram

The block diagram of the I2C module is shown in *Figure 256*.

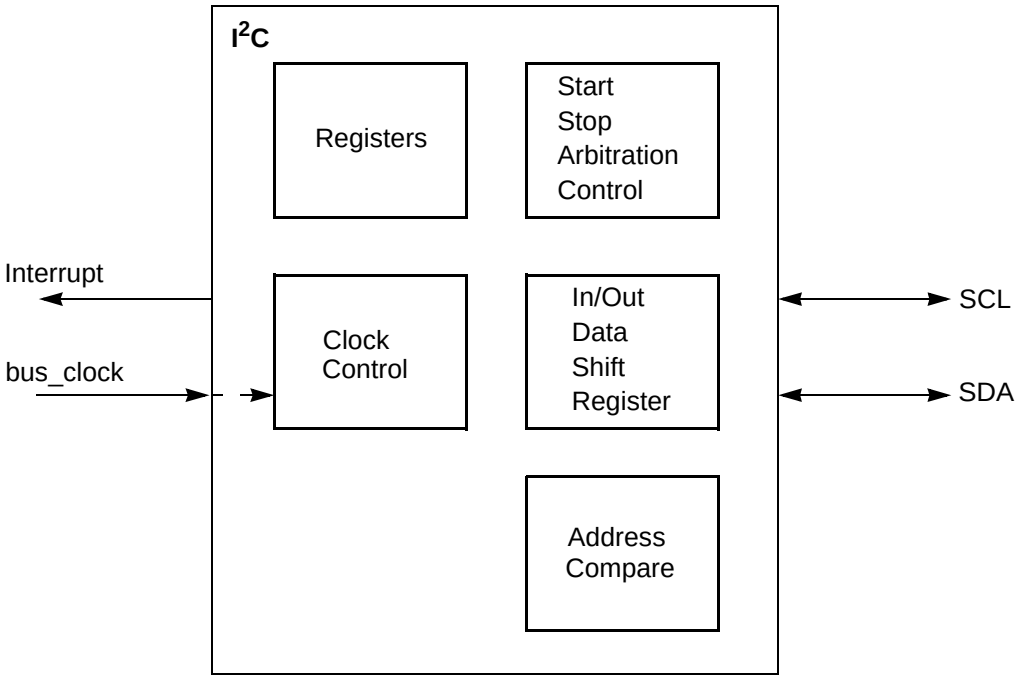


Figure 256. I2C block diagram

25.2 External signal description

The Inter-Integrated Circuit (I2C) module has two external pins, SCL and SDA.

25.2.1 SCL

This is the bidirectional Serial Clock Line (SCL) of the module, compatible with the I2C-Bus specification.

25.2.2 SDA

This is the bidirectional Serial Data line (SDA) of the module, compatible with the I2C-Bus specification.

25.3 Memory map and register description

25.3.1 Module memory map

The memory map for the I2C module is given below in *Table 236*. 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 236. I2C memory map

Base address: 0xFFE3_0000		
Address offset	**Register**	**Location**
0x0	I2C Bus Address Register (IBAD)	on page 582
0x1	I2C Bus Frequency Divider Register (IBFD)	on page 582
0x2	I2C Bus Control Register (IBCR)	on page 589
0x3	I2C Bus Status Register (IBSR)	on page 590
0x4	I2C Bus Data I/O Register (IBDR)	on page 591
0x5	I2C Bus Interrupt Config Register (IBIC)	on page 592

Note: All these registers are accessible only in Supervisor mode.

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.

25.3.2 I2C Bus Address Register (IBAD)

This register contains the address the I2C bus will respond to when addressed as a slave; note that it is not the address sent on the bus during the address transfer.

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

Figure 257. I2C Bus Address Register (IBAD)

Table 237. IBAD field descriptions

Field	Description
ADR	Slave Address. Specific slave address to be used by the I2C Bus module. **Note:** The default mode of I2C Bus is slave mode for an address match on the bus.

25.3.3 I2C Bus Frequency Divider Register (IBFD)

Offset 0x1 Access: Read/write any time

	7	6	5	4	3	2	1	0
R	IBC							
W								
Reset	0	0	0	0	0	0	0	0

Figure 258. I2C Bus Frequency Divider Register (IBFD)

Table 238. 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 239*) 5–3 select the prescaler divider (see *Table 240*) 2–0 select the shift register tap point (see *Table 241*)

Table 239. I-Bus multiplier factor

IBC7–6	MUL
00	01
01	02
10	04
11	RESERVED

Table 240. 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 241. 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 240*. All subsequent tap points are separated by 2^{IBC5-3} as shown in the tap2tap column in *Table 240*. 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.

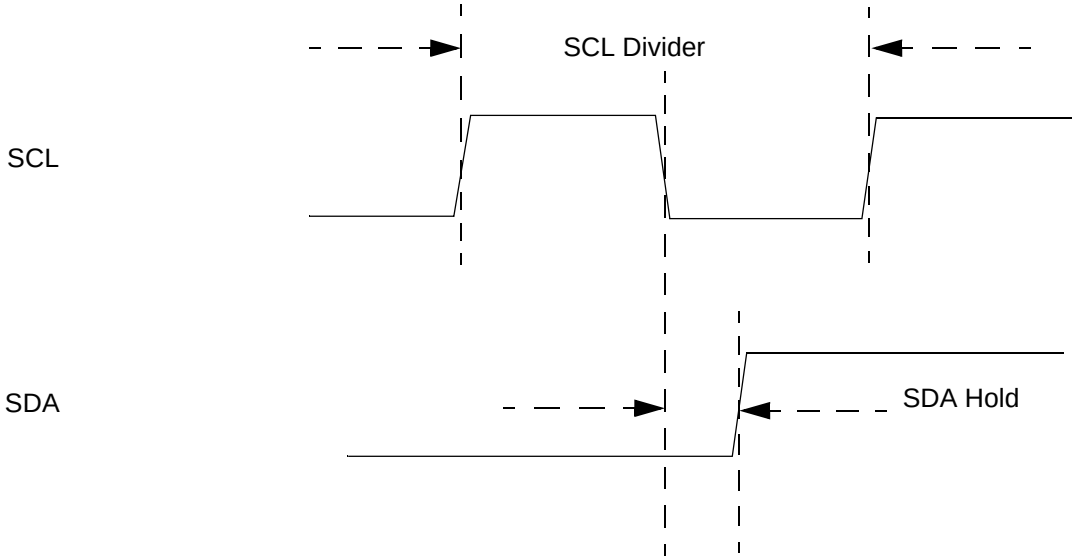


Figure 259. SDA hold time

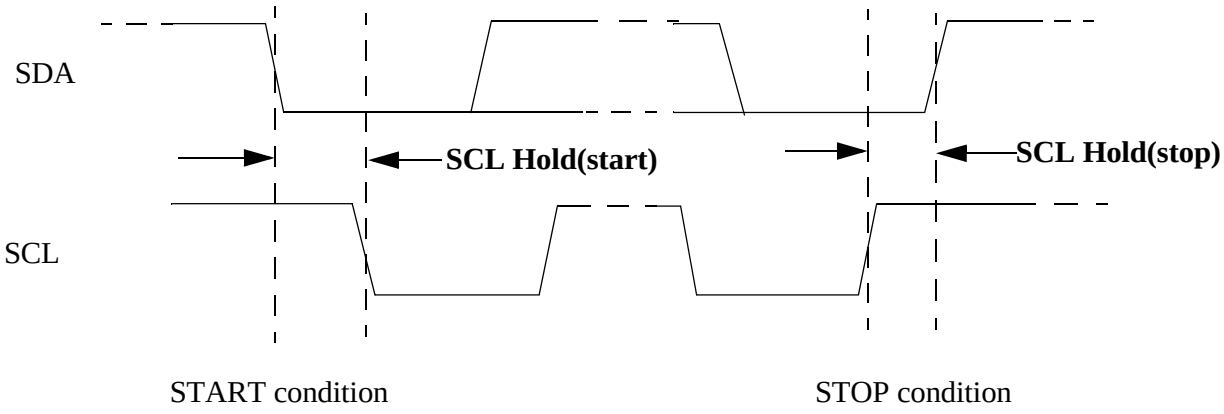


Figure 260. SCL divider and SDA hold

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 242*. 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 242. 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 242. I²C divider and hold values (continued)

	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 242. I^2C divider and hold values (continued)

	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

25.3.4 I2C 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

Figure 261. I2C Bus Control Register (IBCR)

Table 243. IBCR field descriptions

Field	Description
MDIS	Module disable. This bit controls the software reset of the entire I2C 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 I2C Bus module is enabled. This bit must be cleared before any other IBCR bits have any effect **Note:** If the I2C 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 I2C Bus module losing arbitration, after which, bus operation would return to normal.
IBIE	I-Bus Interrupt Enable. 1 Interrupts from the I2C Bus module are enabled. An I2C Bus interrupt occurs provided the IBIF bit in the status register is also set. 0 Interrupts from the I2C 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 I2C 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 I2C 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 243. 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

25.3.5 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

Figure 262. I^2C Bus Status Register (IBSR)

Table 244. 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 244. 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

25.3.6 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

Figure 263. I2C Bus Data I/O Register (IBDR)

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).

25.3.7 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

Figure 264. I^2C Bus Interrupt Config Register (IBIC)

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 245. IBIC field descriptions

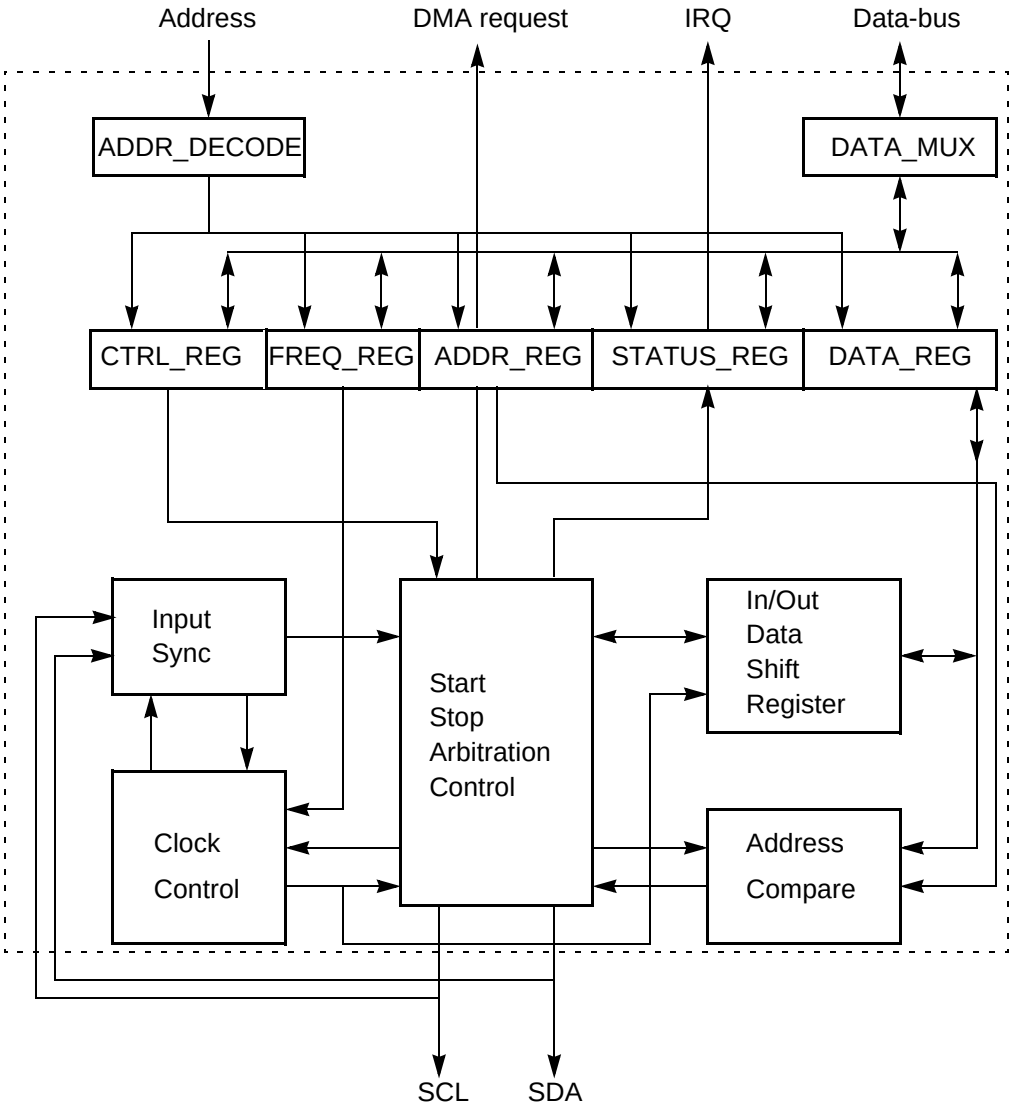
Field	Description
BIIE	Bus Idle Interrupt Enable bit. This config bit can be used to enable the generation of an interrupt once the I^2C bus becomes idle. Once this bit is set, an IBB high-low transition will set the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I^2C bus. 1 Bus Idle Interrupts enabled 0 Bus Idle Interrupts disabled **Note:** This bit cannot be set in the reset state, when the I^2C is in slave mode. It can be set only when the I^2C is in master mode.

25.4 DMA Interface

A simple DMA interface is implemented so that the I^2C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting bit 1 in the Control Register.

The DMA interface is only valid when the I^2C module is configured for Master Mode.

Figure 265. I2C module DMA interface block diagram



At least 3 bytes of data per frame must be transferred from/to the slave when using DMA mode, although in practice it will only be worthwhile using the DMA mode when there is a large number of data bytes to transfer per frame.

Two internal signals, TX request and RX request, are used to signal to a DMA controller when the I2C module requires data to be written or read from the data register.

25.5 Functional description

25.5.1 I-Bus protocol

The I2C 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 266*.

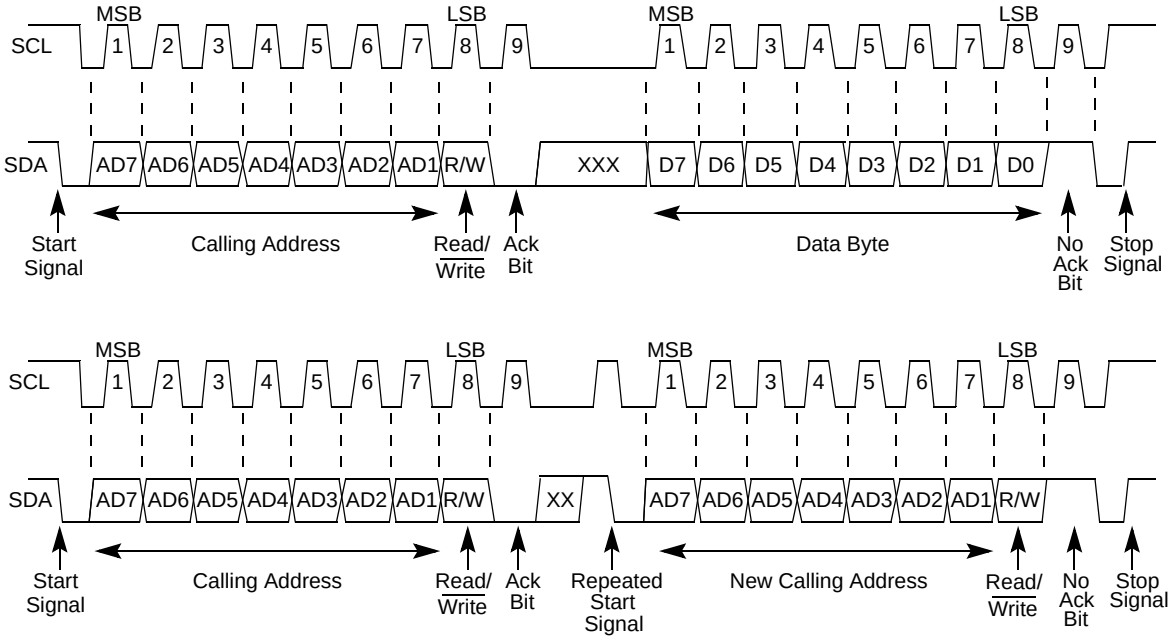


Figure 266. I2C bus transmission signals

25.5.1.1 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 266*, 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.

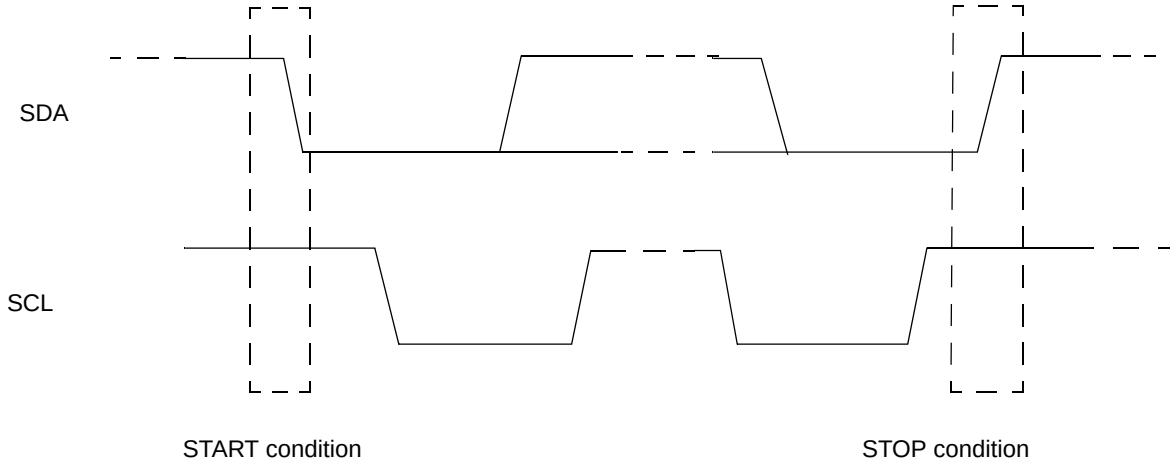


Figure 267. Start and stop conditions

25.5.1.2 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 266*).

No two slaves in the system may have the same address. If the I2C Bus is master, it must not transmit an address that is equal to its own slave address. The I2C Bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle the I2C Bus will revert to slave mode and operate correctly, even if it is being addressed by another master.

25.5.1.3 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 266*. 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.

25.5.1.4 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 266*).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

25.5.1.5 Repeated START signal

As shown in *Figure 266*, 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.

25.5.1.6 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.

25.5.1.7 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 268*). 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.

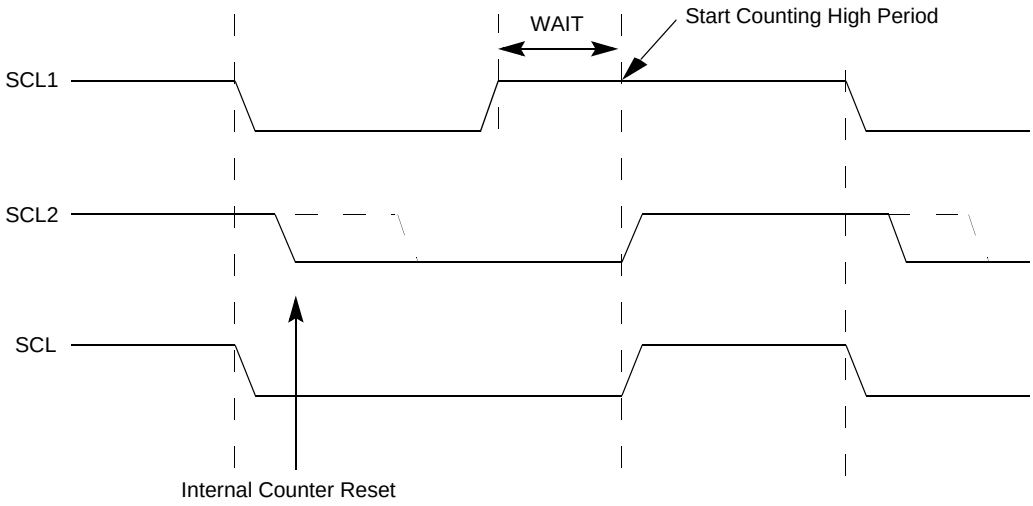


Figure 268. I2C bus clock synchronization

25.5.1.8 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.

25.5.1.9 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.

25.5.2 Interrupts

25.5.2.1 General

The I2C uses only one interrupt vector.

Table 246. Interrupt summary

Interrupt	Offset	Vector	Priority	Source	Description
I2C 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.

25.5.2.2 Interrupt description

There are five types of internal interrupts in the I2C. The interrupt service routine can determine the interrupt type by reading the Status Register.

I2C 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 I2C interrupt is enabled by the IBIE bit in the I2C 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.

25.6 Initialization/application information

25.6.1 I2C programming examples

25.6.1.1 Initialization sequence

Reset will put the I2C 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.

25.6.1.2 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
```

25.6.1.3 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 I^2C 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 I^2C 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 I^2C 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.

25.6.1.4 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. f) 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).

25.6.1.5 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 NOACK bit 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
```

25.6.1.6 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
```

25.6.1.7 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.

25.6.1.8 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.

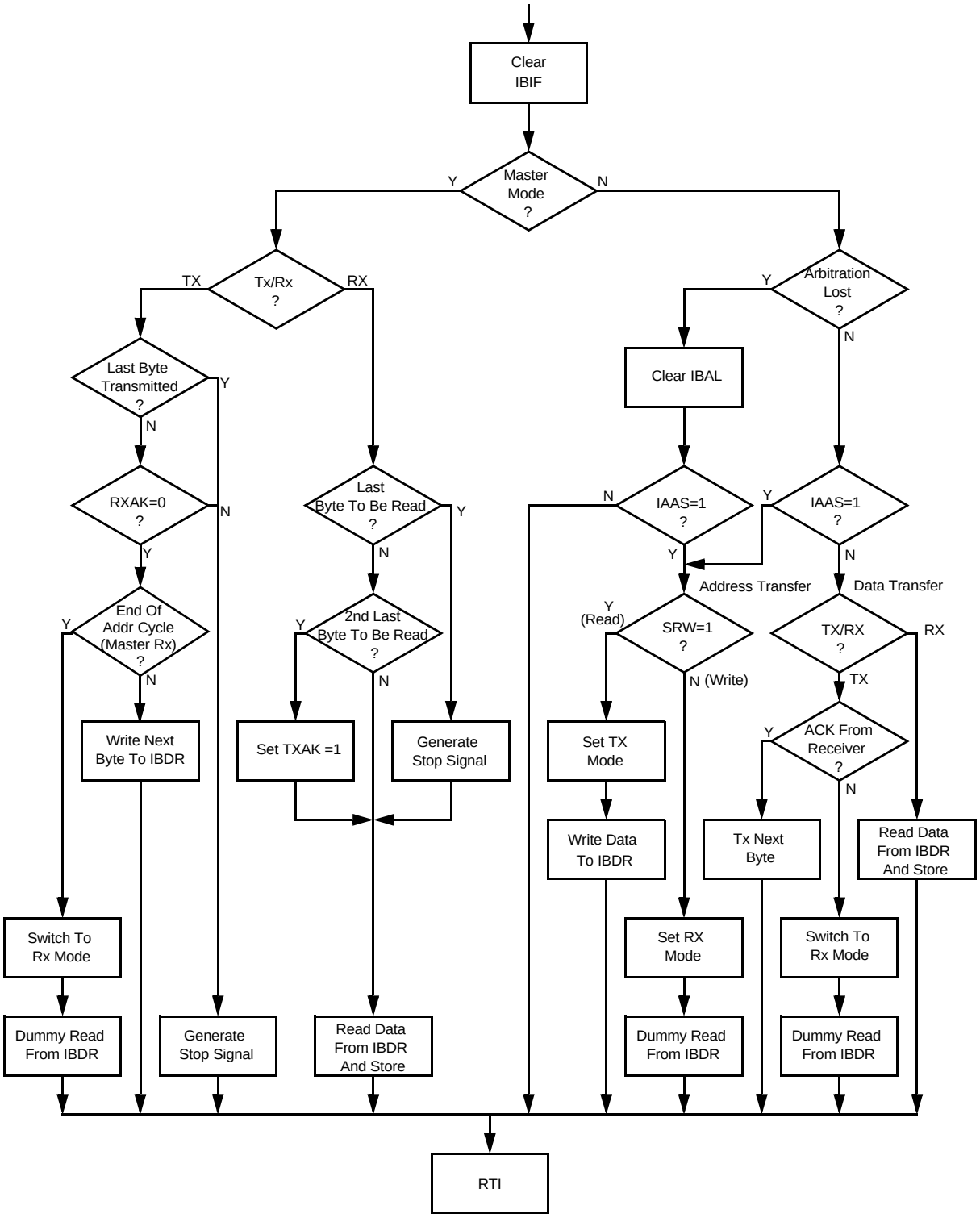


Figure 269. Flow-Chart of Typical I²C Interrupt Routine

25.6.2 DMA application information

The DMA interface on the I^2C is not completely autonomous and requires intervention from the CPU to start and to terminate the frame transfer. DMA mode is valid for master-transmit and master-receive modes only. Software must ensure that the DMA enable bit in the control register is not set when the I^2C module is configured in master mode.

The DMA controller must transfer only one byte of data per Tx/Rx request. This is because there is no FIFO on the I^2C block.

The CPU should also keep the I^2C interrupt enabled during a DMA transfer to detect the arbitration lost condition and take action to recover from this situation. The DMAEN bit in the IBCR register works as a disable for the transfer complete interrupt. This means that during normal transfers (no errors) there always is either an interrupt or a request to the DMA controller, depending on the setting of the DMAEN bit. All error conditions trigger an interrupt and require CPU intervention. The address match condition does not occur in DMA mode as the I^2C should never be configured for slave operation.

The following sections detail how to set up a DMA transfer and what intervention is required from the CPU. It is assumed that the system DMA controller is capable of generating an interrupt after a certain number of DMA transfers have taken place.

25.6.2.1 DMA mode, master transmit

Figure 270 details exactly the operation for using a DMA controller to transmit *n* data bytes to a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the last data byte) can be transferred by the DMA controller. The last data byte must be transferred by the CPU.

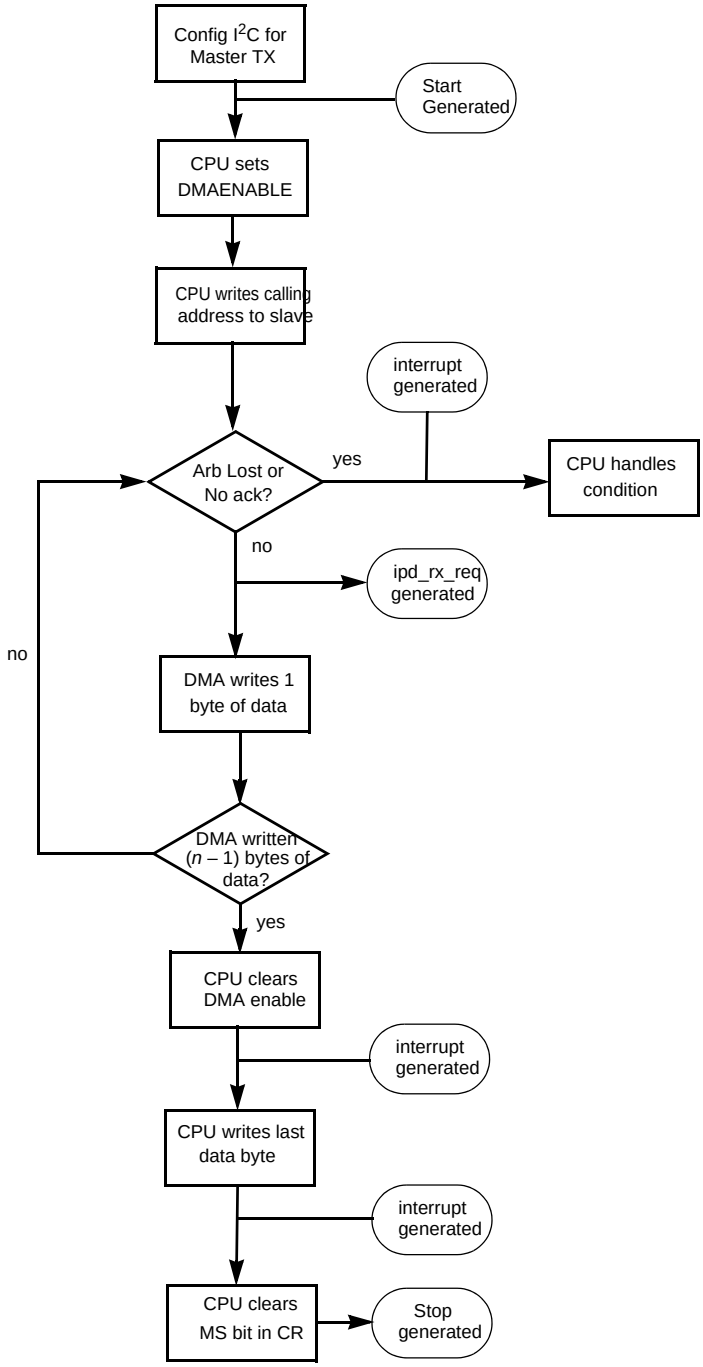


Figure 270. DMA mode master transmit

25.6.2.2 DMA mode, master RX

Figure 271 details the exact operation for using a DMA controller to receive *n* data bytes from a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the two last data bytes) can be read by the DMA controller. The last two data bytes must be transferred by the CPU.

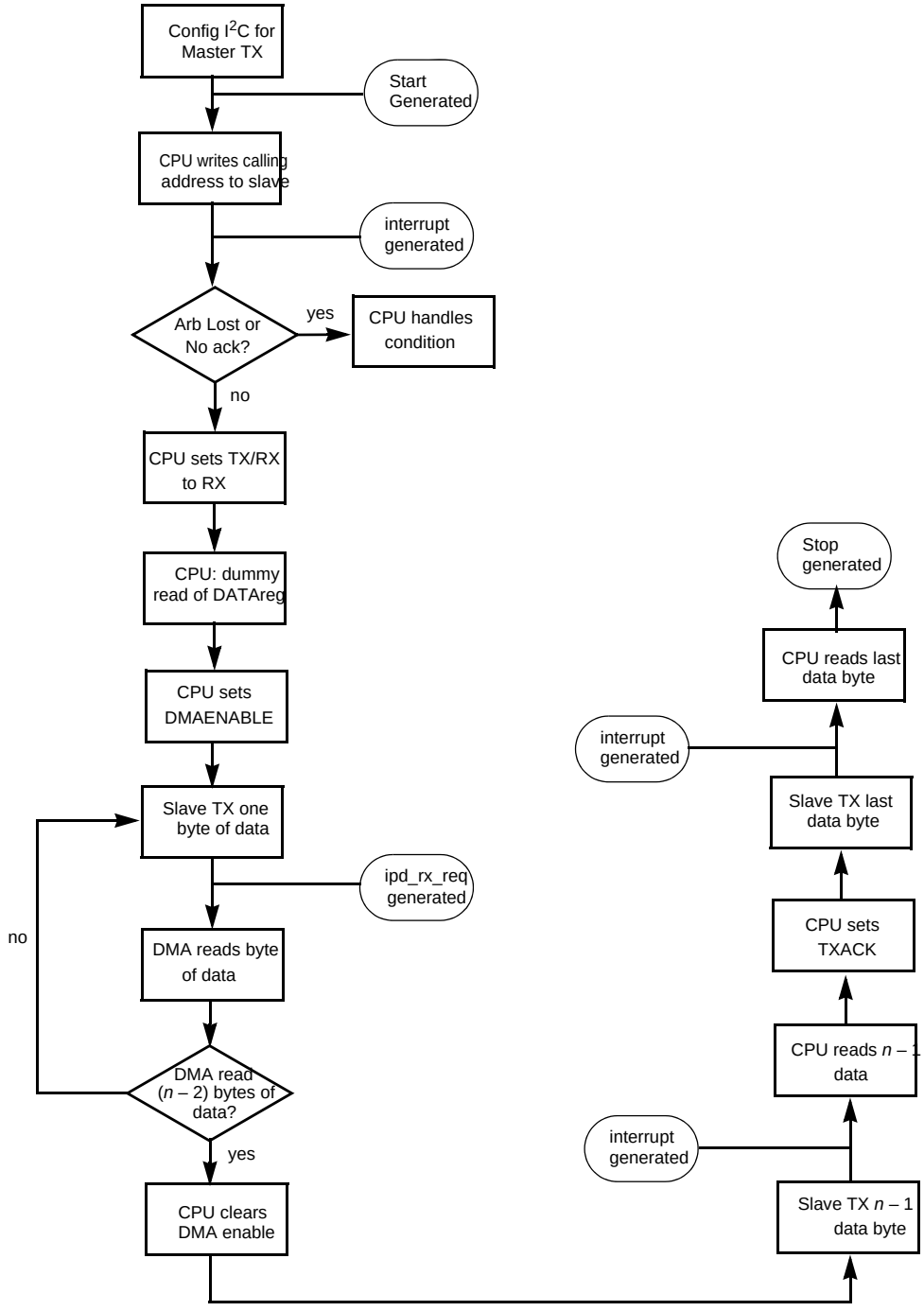


Figure 271. DMA mode master receive

25.6.2.3 Exiting DMA mode, system requirement considerations

As described above, the final transfers of both Tx and Rx transfers need to be managed via interrupt by the CPU. To change from DMA to interrupt driven transfers in the I2C module, disable the DMAEN bit in the IBCR register. The trigger to exit the DMA mode is that the

programmed DMA transfer control descriptor (TCD) has completed all its transfers to/from the I^2C module.

After the last DMA write (TX mode) to the I^2C the module immediately starts the next I^2C-bus transfer. The same is true for RX mode. After the DMA read from the IBDR register the module initiates the next I^2C-bus transfer. This results in two possible scenarios in the DMA mode exiting scheme.

1. Fast reaction

   The DMAEN bit is cleared before the next I^2C-bus transfer completes. In this case, the module raises an interrupt request to the CPU that can be serviced normally.

2. Slow reaction

   The DMAEN bit is cleared after the next I^2C-bus transfer has already completed. In this case, the module does not raise an interrupt request to the CPU. Instead, the TCF bit can be read to determine that the transfer completed and the module is ready for further transfer.

25.6.2.3.1 Fast vs. slow reaction

The reaction time T_R for the system to disable DMAEN after the last DMA controller access to the I^2C is the time required for one byte transfer over the I^2C. In a fast reaction the disabling has to occur before the ninth bit of the data transfer, which is the ACK bit. So the time available is eight times the SCL period.

Equation 7 $T_R = 8 \times T_{SCL}$

In fast mode, with 400 kbit/s, T_{SCL} is 2.5 µs, so T_R is 20 µs.

Depending on the system and DMA controller there are different possibilities for the deassertion of DMAEN. Three options are:

1. CPU intervention via interrupt

   The DMA controller is programmed to signal an interrupt to the CPU, which is then responsible for the deassertion of DMAEN. This scheme is supported by most systems but can result in a slow reaction time if higher priority interrupts interfere. Therefore, the interrupt handling routine can become complicated as it has to check which of the two scenarios happened (check TCF bit) and act accordingly. In case of slow reaction, you can force an interrupt for the I^2C in the interrupt controller to have the further transfer handled by the normal I^2C interrupt routine. The use of nested interrupts can cause problems in this scenario, if the DMA interrupt stalls between the deassertion and the DMAEN bit and the checking of the TCF bit.

2. DMA channel linking (if supported)

   The transfer control descriptor in the DMA controller that performs the data transfer is linked to another channel that does a write to the I^2C IBCR register to disable the DMAEN bit. This is probably the fastest system solution, but it uses two DMA channels. On the system level, no higher priority DMA requests must occur between the two linked TCDs because those can result in slow reaction.

3. DMA scatter/gather process (if supported)

   The transfer control descriptor in the DMA controller that performs the data transfer has the scatter-gather feature activated. This feature initiates a reload of another TCD from system RAM after the completion of the first TCD. The new TCD has its start bit already set and immediately starts the required write to the I^2C IBCR register to disable the DMAEN bit. This TCD also has scatter-gather activated and is programmed to reload the initial TCD upon completion, bringing the system back into a ready-for-I^2C-transfer

state. The advantage over the two other solutions is that this does not require CPU intervention or a second DMA channel. This comes at the cost of 64 bytes RAM (two TCDs), some system bus transfer overhead, and a little increase in application code complexity. On the system level, no higher priority DMA requests must occur during the scatter-gather process because those can result in a slow reaction.

Example latencies for a 32 MHz system with a full speed 32-bit AHB bus and an I^2C connected via half speed IPI bus:

- Accessing the I^2C from the DMA controller via IPI bus typically requires four cycles (consecutive accesses to the I^2C could be faster):

Equation 8 $4 \times T_{IPI} = 4 / 16$ MHz = 250 ns

- Reloading a new TCD (8×32-bit) via AHB to the DMA controller (scatter/gather process):

Equation 9 $8 \times T_{AHB} = 8 / 32$ MHz = 250 ns

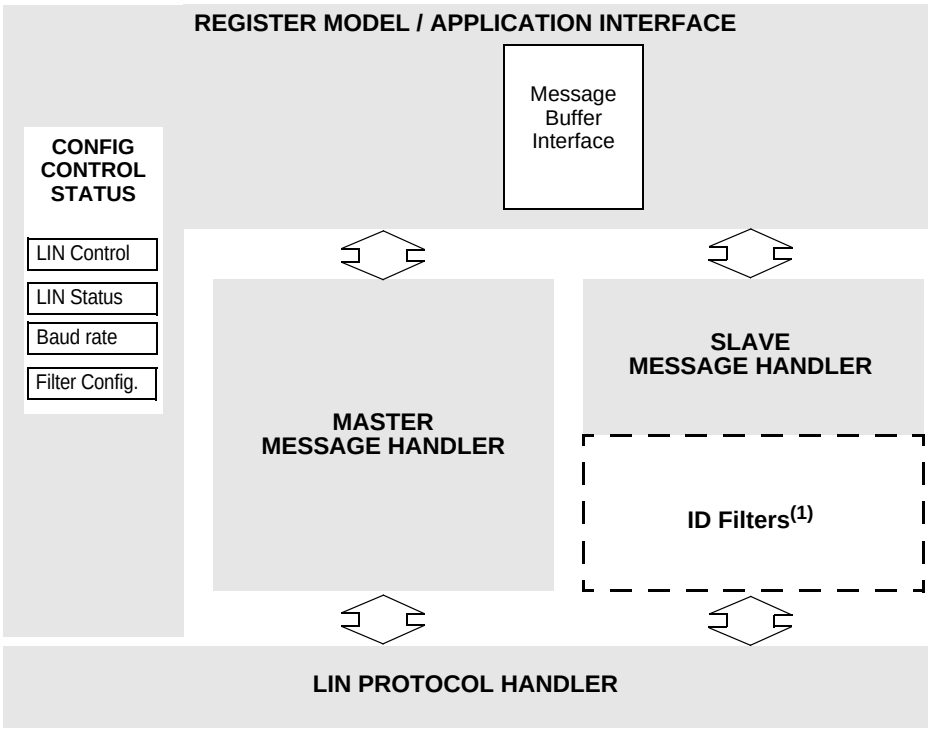
With the DMA scatter-gather process, the required IBCR access can be done in 0.5 µs, leaving a large margin of 19.5 µs for additional system delays. The slow reaction case can be prevented in this way. The system user must decide which usage model suits his overall requirements best.

26 LIN Controller (LINFlexD)

26.1 Introduction

The LINFlexD (Local Interconnect Network Flexible with DMA support) controller interfaces the LIN network and supports the LIN protocol versions 1.3, 2.0, 2.1 and J2602 in both Master and Slave modes. LINFlexD 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.

Figure 272 shows the LINFlexD block diagram.



1 Filter activation optional

Figure 272. LINFlexD block diagram

26.2 Main features

The LINFlexD controller can operate in several modes, each of which has a distinct set of features. These distinct features are described in the following sections.

In addition, the LINFlexD controller has several features common to all modes:

- 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

26.2.1 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 purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with as clock source
- Identifier filters for autonomous message handling in Slave mode

26.2.2 UART mode features

- Full-duplex communication
- Selectable frame size:
  - 8-bit frame
  - 9-bit frame
  - 16-bit frame
  - 17-bit frame
- Selectable parity:
  - Even
  - Odd
  - 0
  - 1
- 4-byte buffer for reception, 4-byte buffer for transmission
- 12-bit counter for timeout management

26.3 The LIN protocol

The LIN (Local Interconnect Network) is a serial communication protocol. The topology of a LIN network is shown in *Figure 273*. A LIN network consists of:

- One master
- Several slave
- The LIN bus

A master node contains the master task as well as a slave task, all other nodes contain a slave task only. The master node decides when and which frame shall be transferred on the bus. The slave task provides the data to be transported by the frame.

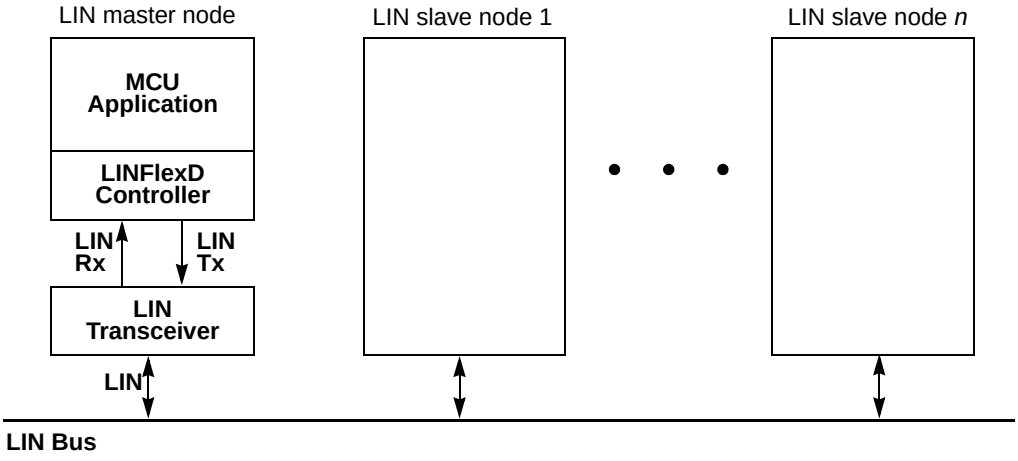


Figure 273. LIN network topology

26.3.1 Dominant and recessive logic levels

The LIN bus defines two logic levels, “dominant” and “recessive”, as follows:

- Dominant: logical low level (0)
- Recessive: logical high level (1)

26.3.2 LIN frames

A frame consists of a header provided by the master task and a response provided by the slave task, as shown in *Figure 274*.

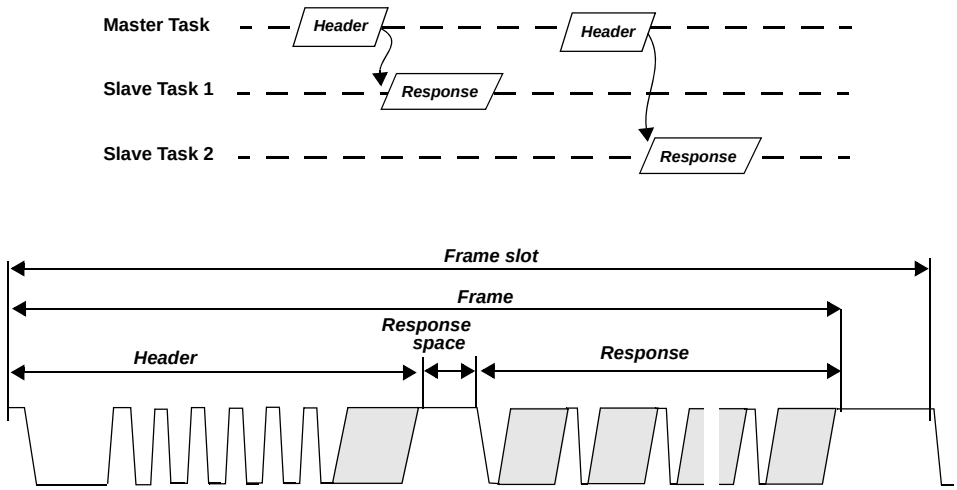


Figure 274. LIN frame structure

26.3.3 LIN header

The header consists of:

- A break field (described in *Section 26.3.3.1: Break field*)
- A sync (described in *Section 26.3.3.2: Sync*)
- An identifier (described in *Section 26.3.4.2: Identifier*)

The slave task associated with the identifier provides the response.

26.3.3.1 Break field

The break field, shown in *Figure 275*, is used to signal the beginning of a new frame. It is always generated by the master and consists of:

- At least 13 dominant bits including the start bit
- At least one recessive bit that functions as break delimiter



Figure 275. Break field

26.3.3.2 Sync

The sync pattern is a byte consisting of alternating dominant and recessive bits as shown in *Figure 276*. It forms a data value of 0x55.

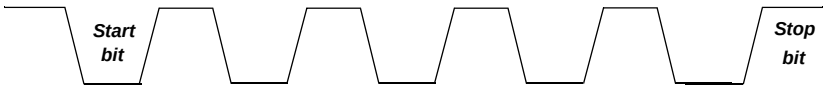


Figure 276. Sync pattern

26.3.4 Response

The response consists of:

- A data field (described in *Section 26.3.4.1: Data field*)
- A checksum (described in *Section 26.3.4.3: Checksum*)

The slave task interested in the data associated with the identifier receives the response and verifies the checksum.

26.3.4.1 Data field

The structure of the data field transmitted on the LIN bus is shown in *Figure 277*. The LSB of the data is sent first and the MSB last. The start bit is encoded as a dominant bit and the stop bit is encoded as a recessive bit.

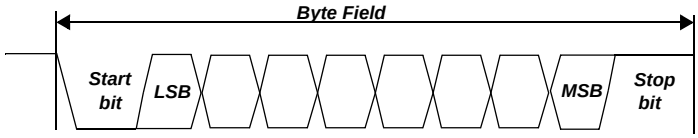


Figure 277. Structure of the data field

26.3.4.2 Identifier

The identifier, shown in *Figure 278*, consists of two sub-fields:

- The identifier value (in bits 0–5)
- The identifier parity (in bits 6–7)

The parity bits P0 and P1 are defined as follows:

- P0 = ID0 xor ID1 xor ID2 xor ID4
- P1 = not(ID1 xor ID3 xor ID4 xor ID5)

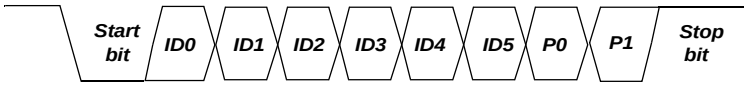


Figure 278. Identifier

26.3.4.3 Checksum

The checksum contains the inverted 8-bit sum (with carry) over one of two possible groups:

- The classic checksum sums all data bytes, and is used for communication with LIN 1.3 slaves.
- The enhanced checksum sums all data bytes and the identifier, and is used for communication with LIN 2.0 (or later) slaves.

26.4 LINFlexD and software intervention

The increasing number of communication peripherals embedded on microcontrollers (for example, CAN, LIN, SPI) requires more and more CPU resources for the 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 is usually the case.

To minimize the CPU load in Master mode, LINFlexD handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlexD does not request any software (that is, application) 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, LINFlexD 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, LINFlexD 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.

26.5 Summary of operating modes

The LINFlexD controller has three operating modes:

- Normal
- Initialization
- Sleep

After a hardware reset, the LINFlexD controller is in Sleep mode to reduce power consumption.

The transitions between these modes are shown in *Figure 279*. The software instructs LINFlexD to enter Initialization mode or Sleep mode by setting LINCR1[INIT] or LINCR1[SLEEP], respectively.

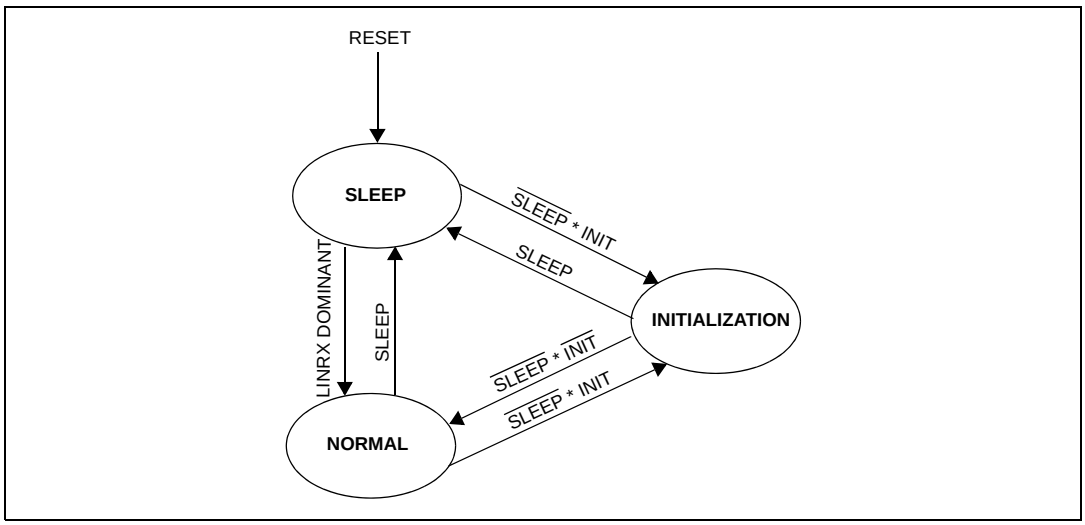


Figure 279. LINFlexD controller operating modes

In addition to these controller-level operating modes, the LINFlexD controller also supports several protocol-level modes:

- LIN mode:
  - Master mode
  - Slave mode
  - Slave mode with identifier filtering
  - Slave mode with automatic resynchronization
- UART mode
- Test modes:
  - Loop Back mode
  - Self Test mode

These modes are discussed in detail in subsequent sections.

26.6 Controller-level operating modes

26.6.1 Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter or exit this mode, the software sets or clears LINCR1[INIT], respectively.

In Initialization mode, all message transfers to and from the LIN bus are stopped and the LIN bus output (LINTX) is recessive.

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlexD controller, the software must:

- Select the desired mode (Master, Slave or UART)
- Set up the baud rate register
- If LIN Slave mode with filter activation is selected, initialize the identifier list

26.6.2 Normal mode

After initialization is complete, the software must clear LINCR1[INIT] to put the LINFlexD controller into Normal mode.

26.6.3 Sleep (low-power) mode

To reduce power consumption, LINFlexD has a low-power mode called Sleep mode. In this mode, the LINFlexD clock is stopped. Consequently, the LINFlexD will not update the status bits, but software can still access the LINFlexD registers.

To enter this mode, the software must set LINCR1[SLEEP].

LINFlexD can be awakened (exit Sleep mode) in one of two ways:

- The software clears LINCR1[SLEEP]
- Automatic wake-up is enabled (LINCR1[AWUM] is set) and LINFlexD detects LIN bus activity (that is, if a wakeup pulse of 150 µs is detected on the LIN bus)

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing LINCR1[SLEEP] if LINCR1[AWUM] is set. To exit from Sleep mode if LINCR1[AWUM] is cleared, the software must clear LINCR1[SLEEP] when a wake-up event occurs.

26.7 LIN modes

26.7.1 Master mode

In Master mode, the software uses the message buffer to handle the LIN messages.

Master mode is selected when LINCR1[MME] is set.

26.7.1.1 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 LINFlexD the application must set up the identifier, the data field length and configure the message (direction and checksum type) in the BIDR register before requesting the header transmission by setting LINCR2[HTRQ].

26.7.1.2 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 software must provide the data to LINFlexD before requesting the header transmission. The software stores the data in the message buffer BDR. According to

the data field length LINFlexD transmits the data and the checksum. The software uses the BIDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the software sets this bit the response is sent by LINFlexD (publisher). Clearing this bit configures the message buffer as subscriber.

26.7.1.3 Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlexD stores the data received from the slave in the message buffer and stores the message status in the LINSR.

26.7.1.4 Error detection and handling

LINFlexD 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.

Table 247 lists the errors detected in Master mode and the LINFlexD controller's response to these errors.

Table 247. Errors in Master mode

Error	Description	LINFlexD response to error
Bit error	During transmission, the value read back from the bus differs from the transmitted value	– Stops the transmission of the frame after the corrupted bit – Generates an interrupt if LINIER[BEIE] is set – Returns to idle state
Framing error	A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field)	If encountered during reception: – Discards the current frame – Generates an interrupt if LINIER[FEIE] is set – Returns immediately to idle state
Checksum error	The computed checksum does not match the received checksum	If encountered during reception: – Discards the current frame – Generates an interrupt if LINIER[CEIE] is set – Returns to idle state
Response and frame timeout	Refer to *Section 26.12.1: 8-bit timeout counter*, for more details	

26.7.1.5 Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting the BOF bit in the LINESR.Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer is overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.

26.7.2 Slave mode

In Slave mode the software uses the message buffer to handle the LIN messages.

Slave mode is selected when the LINCR1[MME] is cleared.

26.7.2.1 Data transmission (transceiver as publisher)

When LINFlexD receives the identifier, an RX interrupt is generated. The software must:

- Read the received ID in the BIDR register
- Fill the BDR registers
- Specify the data field length using the BIDR[DFL] field
- Trigger the data transmission by setting LINCR2[DTRQ]

One or several identifier filters can be configured for transmission by setting the DIR bits in the corresponding IFCR registers and activated by setting one or several bits in the IFER register.

When at least one identifier filter is configured in transmission and activated, and if the received ID matches the filter, a specific TX interrupt is generated.

Typically, the software has to copy the data from RAM locations to the BDRL and BDRM registers. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter which matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer which points to the right data array in the RAM area and copy this data to the BDRL and BDRM registers (see *Figure 281*).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BDIR register. The software fills the BDRL and BDRM registers and triggers the data transmission by setting LINCR2[DTRQ].

If LINFlexD 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 (refer to *Section 26.7.3: Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

26.7.2.2 Data reception (transceiver as subscriber)

When LINFlexD receives the identifier, an RX interrupt is generated. The software must:

- Read the received ID in the BIDR register
- Specify the data field length using the BIDR[DFL] field before the reception of 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 clearing the DIR bit in the corresponding IFCR registers and activated by clearing one or several bits in the IFER register.

When at least one identifier filter is configured in reception and activated, and if the received ID matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the software has to copy the data from the BDRL and BDRM registers to RAM locations. To copy the data to the right location, the software has to identify the data by means

of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter which matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer which points to the right data array in the RAM area and copy this data from the BDRL and BDRM registers to the RAM (see *Figure 281*).

Using a filter avoids the software reading the ID value in the BIDR register, and configuring the direction, the data field length and the checksum type in the BIDR register.

If LINFlexD 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 (refer to *Section 26.7.3: Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

26.7.2.3 Data discard

When LINFlexD receives the identifier, an RX interrupt is generated. If the received identifier does not concern the node, the software must set LINCR2[DDRQ]. LINFlexD returns to idle state.

26.7.2.4 Error detection and handling

Table 248 lists the errors detected in Slave mode and the LINFlexD controller's response to these errors.

Table 248. Errors in Slave mode

Error	Description	LINFlexD response to error
Bit error	During transmission, the value read back from the bus differs from the transmitted value	– Stops the transmission of the frame after the corrupted bit – Generates an interrupt if LINIER[BEIE] is set – Returns to idle state
Framing error	A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field)	If encountered during reception: – Discards the current frame – Generates an interrupt if LINIER[FEIE] is set – Returns immediately to idle state
Checksum error	The computed checksum does not match the received checksum	If encountered during reception: – Discards the received frame – Generates an interrupt if LINIER[CEIE] is set – Returns to idle state
Header error	An error occurred during header reception (break delimiter error, inconsistent sync field, header timeout)	If encountered during header reception, a break field error, an inconsistent sync field, or a timeout: – Discards the header – Generates an interrupt if LINIER[HEIE] is set – Returns to idle state

26.7.2.5 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.

26.7.2.6 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.

26.7.2.7 Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The LINFlex controller signals the overrun condition by setting the BOF bit in the LINESR (LINESR[BOF]).

Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer will be overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.
- If buffer is not released (LINSR[RMB] = 1) before reception of next Identifier and if RBLM is set then ID along with the data is discarded.

26.7.3 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. When a slave node receives a header, it 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 LINFlexD controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources which would otherwise be needed by software for filtering.

The filtering is accomplished through the use of IFCR registers. These registers have the names IFCR0 through IFCR15. This section also uses the nomenclature $IFCR_{2n}$ and $IFCR_{2n+1}$; in this nomenclature, n is an integer, and the corresponding IFCR register is calculated using the formula in the subscript.

26.7.3.1 Filter submodes

Usually each of the 16 IFCRs is used to filter one dedicated identifier, but this means that the LINFlexD controller could filter a maximum of 16 identifiers. In order to be able to handle more identifiers, the filters can be configured to operate as masks.

Table 249 describes the two available filter submodes.

Table 249. Filter submodes

Submode	Description
Identifier list	Both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register. This is the default submode for the LINFlexD controller.
Mask	The identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.

The bit mapping and register organization in these two submodes is shown in *Figure 280*.

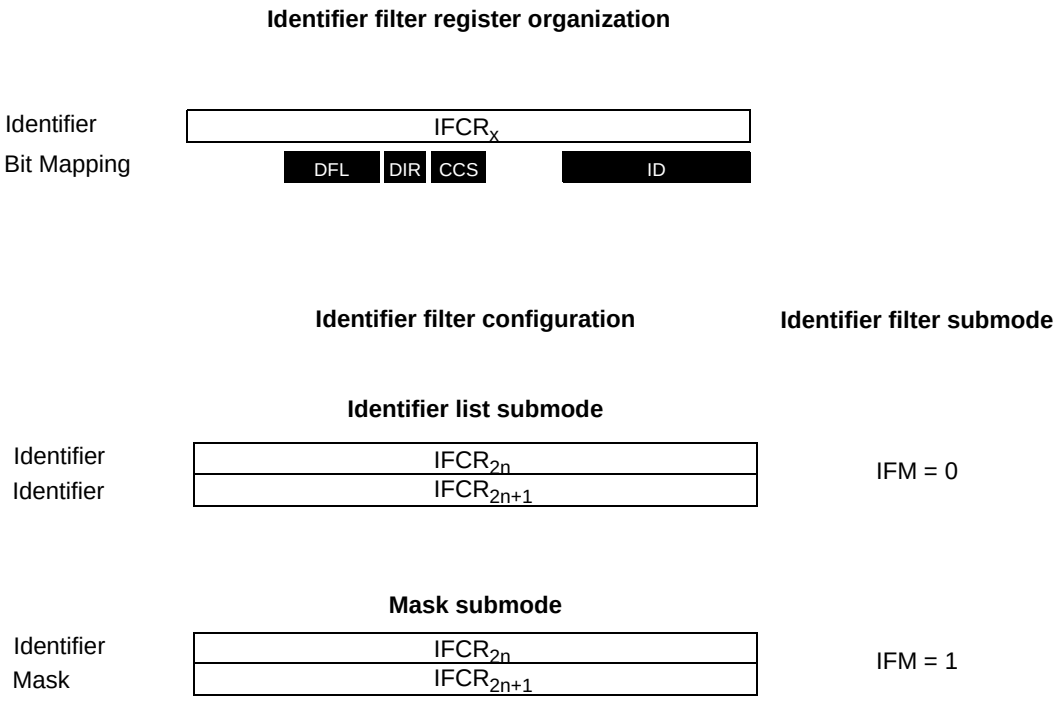


Figure 280. Filter configuration - register organization

26.7.3.2 Identifier filter submode configuration

The identifier filters are configured in the IFCR registers. To configure an identifier filter the filter must first be activated by setting the corresponding bit in the IFER[FACT] field. The submode (identifier list or mask) for the corresponding IFCR register is configured by the IFMR[IFM] field. For each filter, the IFCR register is used to configure:

- The ID or mask
- The direction (TX or RX)
- The data field length
- 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.

Further details are provided in *Table 250* and *Figure 281*.

Table 250. 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 IDs
a (a > 0)	a	0	– TX interrupt on IDs matching the filters, – RX interrupt on all other IDs 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 IDs matching the TX filters, – RX interrupt on IDs matching the RX filters, – All other IDs discarded (no interrupt)
b (b > 0)	0	b	– RX interrupt on IDs matching the filters, – TX interrupt on all other IDs if BF bit is set, no TX interrupt if BF bit is reset

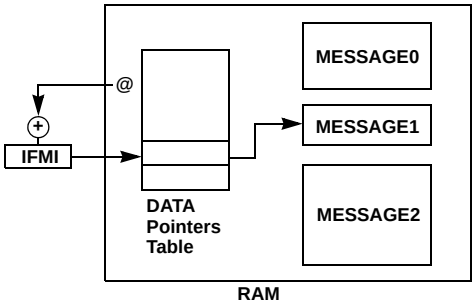


Figure 281. Identifier match index

26.7.4 Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in Slave mode if $f_{ipg_clock_lin}$ tolerance is greater than 1.5%. This feature compensates a deviation up to 14%, as specified in the LIN standard.

This mode is similar to Slave mode as described in *Section 26.7.2: Slave mode*, with the addition of automatic resynchronization enabled by the LINCR1[LASE] bit. In this mode LINFlexD adjusts the fractional baud rate generator after each synch field reception.

26.7.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN break, the time duration between five falling edges on RDI is sampled on as shown in *Figure 282*. Then the LFDIV value (and its associated LINIBRR and LINFBRR registers) is automatically updated at the end of the fifth falling edge. During LIN sync field measurement, the LINFlexD state machine is stopped and no data is transferred to the data register.

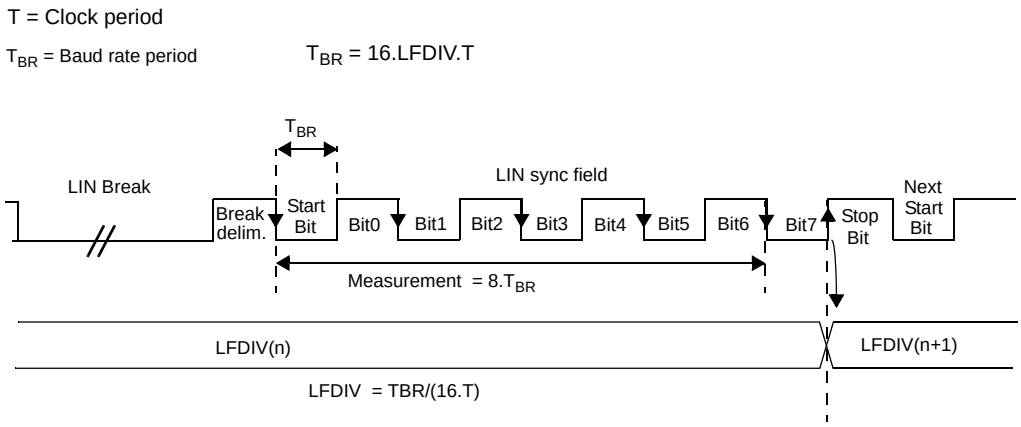


Figure 282. LIN sync field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 20 bits in the LINIBRR register and the fraction is coded on 4 bits in the LINFBRR register.

If LINCR1[LASE] is set, LFDIV is automatically updated at the end of each LIN sync field.

Three registers are used internally to manage the auto-update of the LINFlexD 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.

26.7.4.2 Deviation error on the sync field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN sync 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 sync 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 LINFlexD_RX pin the $f_{ipg_clock_lin}$ clock.

The second check is based on a measurement of time between each falling edge of the sync 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 LINFlexD_RX pin the $f_{ipg_clock_lin}$ clock.

Note that the LINFlexD 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.

26.8 Test modes

The LINFlexD controller includes two test modes, Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1 register. These bits must be configured while LINFlexD is in Initialization mode. After one of the two test modes has been selected, LINFlexD must be started in Normal mode.

26.8.1 Loop Back mode

LINFlexD can be put in Loop Back mode by setting LINCR1[LBKM]. In Loop Back mode, the LINFlexD treats its own transmitted messages as received messages. This is illustrated in *Figure 283*.

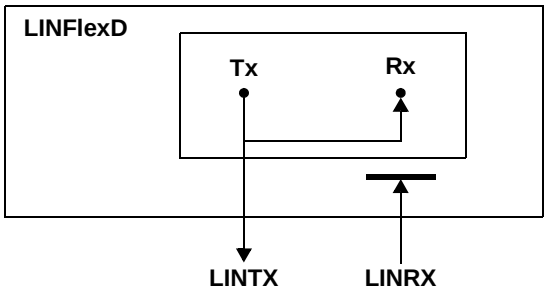


Figure 283. LINFlexD in Loop Back mode

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 LINFlexD performs an internal feedback from its Tx output to its Rx input. The actual value of the **LINRX** input pin is disregarded by the LINFlexD. The transmitted messages can be monitored on the **LINTX** pin.

26.8.2 Self Test mode

LINFlexD can be put in Self Test mode by setting LINCR1[LBKM] and LINCR1[SFTM]. This mode can be used for a “Hot Self Test”, meaning the LINFlexD 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 LINFlexD and the LINTX pin is held recessive. This is illustrated in *Figure 284*.

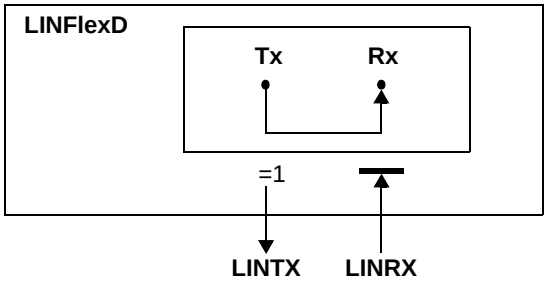


Figure 284. LINFlexD in Self Test mode

26.9 UART mode

The main features of UART mode are presented in *Section 26.2.2: UART mode features*.

26.9.1 Data frame structure

26.9.1.1 8-bit data frame

The 8-bit UART data frame is shown in *Figure 285*. The 8th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. 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.

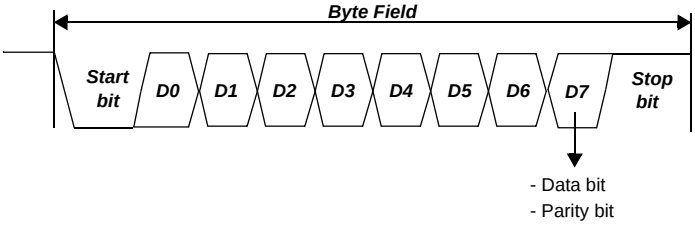


Figure 285. UART mode 8-bit data frame

26.9.1.2 9-bit data frame

The 9-bit UART data frame is shown in *Figure 286*. The 9th bit is a parity bit. Parity (even, odd, 0, or 1) can be selected by the by the UARTCR[PC] field. 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. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

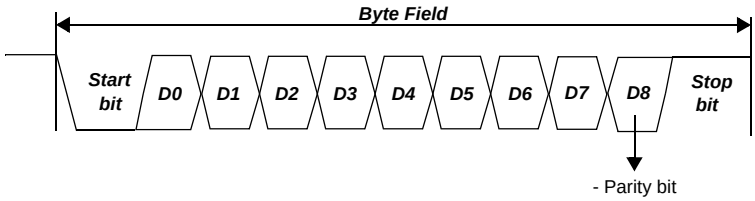


Figure 286. UART mode 9-bit data frame

26.9.1.3 16-bit data frame

The 16-bit UART data frame is shown in *Figure 287*. The 16th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

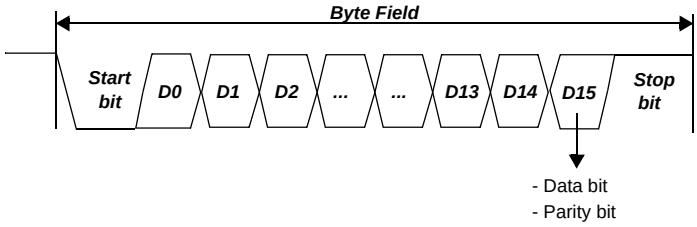


Figure 287. UART mode 16-bit data frame

26.9.1.4 17-bit data frame

The 17-bit UART data frame is shown in *Figure 288*. The 17th bit is the parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

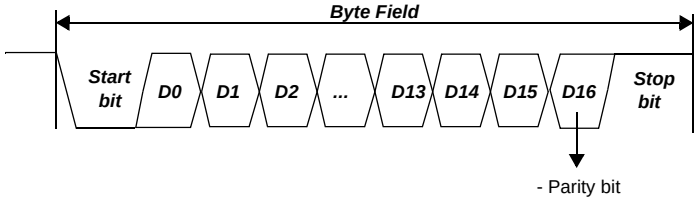


Figure 288. UART mode 17-bit data frame

26.9.2 Buffer

The 8-byte buffer is divided into two parts — one for receiver and one for transmitter — as shown in *Table 251*.

Table 251. UART buffer structure

BDR	UART mode
0	Tx0
1	Tx1
2	Tx2
3	Tx3
4	Rx0
5	Rx1
6	Rx2
7	Rx3

For 16-bit frames, the lower 8 bits will be written in BDR0 and the upper 8 bits will be written in BDR1.

26.9.3 UART transmitter

In order to start transmission in UART mode, the UARTCR[UART] and UARTCR[TXEN] bits must be set. Transmission starts when BDR0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by the UARTCR[TDFLTFC] field (see *Table 264*).

The Transmit buffer size is as follows:

- 4 bytes when UARTCR[WL[1]] = 0
- 2 half-words when UARTCR[WL[1]] = 1

Therefore, the maximum transmission that can be triggered is 4 bytes (2 half-words). After the programmed number of bytes has been transmitted, the UARTSR[DTFTFF] flag is set. If the UARTCR[TXEN] field is cleared during a transmission, the current transmission is completed, but no further transmission can be invoked. The buffer can be configured in FIFO mode (mandatory when DMA Tx is enabled) by setting UARTCR[TFBM].

The access to the BDRL register is shown in *Table 252*.

Table 252. BDRL access in UART mode

Access	Mode(1)	Word length(2)	IPS operation result
Write Byte0	FIFO	Byte	OK
Write Byte1-2-3	FIFO	Byte	IPS transfer error
Write Half-word0-1	FIFO	Byte	IPS transfer error
Write Word	FIFO	Byte	IPS transfer error
Write Byte0-1-2-3	FIFO	Half-word	IPS transfer error
Write Half-word0	FIFO	Half-word	OK
Write Half-word1	FIFO	Half-word	IPS transfer error
Write Word	FIFO	Half-word	IPS transfer error
Read Byte0-1-2-3	FIFO	Byte/Half-word	IPS transfer error

Table 252. BDRL access in UART mode (continued)

Access	Mode(1)	Word length(2)	IPS operation result
Read Half-word0-1	FIFO	Byte/Half-word	IPS transfer error
Read Word	FIFO	Byte/Half-word	IPS transfer error
Write Byte0-1-2-3	BUFFER	Byte/Half-word	OK
Write Half-word0-1	BUFFER	Byte/Half-word	OK
Write Word	BUFFER	Byte/Half-word	OK
Read Byte0-1-2-3	BUFFER	Byte/Half-word	OK
Read Half-word0-1	BUFFER	Byte/Half-word	OK
Read Word	BUFFER	Byte/Half-word	OK

1. As specified by UARTCR[TFBM]
2. As specified by the WL[1] and WL[0] bits of the UARTCR register. In UART FIFO mode (UARTCR[TFBM] = 1),any read operation causes an IPS transfer error.

26.9.4 UART receiver

Reception of a data byte is started as soon as the software completes the following tasks in order:

1. Exits Initialization mode
2. Sets the UARTCR[RXEN] field
3. Detects the start bit

There is a dedicated data buffer for received data bytes. Its size is as follows:

- 4 bytes when UARTCR[WL[1]] = 0
- 2 half-words when UARTCR[WL[1]] = 1

After the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRFRFE] field is set. If the UARTCR[RXEN] field is cleared during a reception, the current reception is completed, but no further reception can be invoked until UARTCR[RXEN] is set again.

The buffer can be configured in FIFO mode (required when DMA Rx is enabled) by setting UARTCR[RFBM].

The access to the BDRM register is shown in *Table 253*.

Table 253. BDRM access in UART mode

Access	Mode(1)	Word length(2)	IPS operation result
Read Byte4	FIFO	Byte	OK
Read Byte5-6-7	FIFO	Byte	IPS transfer error
Read Half-word2-3	FIFO	Byte	IPS transfer error
Read Word	FIFO	Byte	IPS transfer error
Read Byte4-5-6-7	FIFO	Half-word	IPS transfer error
Read Half-word2	FIFO	Half-word	OK
Read Half-word3	FIFO	Half-word	IPS transfer error

Table 253. BDRM access in UART mode (continued)

Access	Mode(1)	Word length(2)	IPS operation result
Read Word	FIFO	Half-word	IPS transfer error
Write Byte4-5-6-7	FIFO	Byte/Half-word	IPS transfer error
Write Half-word2-3	FIFO	Byte/Half-word	IPS transfer error
Write Word	FIFO	Byte/Half-word	IPS transfer error
Read Byte4-5-6-7	BUFFER	Byte/Half-word	OK
Read Half-word2-3	BUFFER	Byte/Half-word	OK
Read Word	BUFFER	Byte/Half-word	OK
Write Byte4-5-6-7	BUFFER	Byte/Half-word	IPS transfer error
Write Half-word2-3	BUFFER	Byte/Half-word	IPS transfer error
Write Word	BUFFER	Byte/Half-word	IPS transfer error

1. As specified by UARTCR[RFBM]
2. As specified by the WL[1] and WL[0] bits of the UARTCR register

Table 254 lists some common scenarios, controller responses, and suggestions when the LINFlexD controller is acting as a UART receiver.

Table 254. UART receiver scenarios

Scenario	Responses and suggestions
The software does not know (in advance) how many bytes will be received.	Do not program UARTCR[RDFLRFC] in advance. When this field is zero (as it is after reset), reception occurs on a byte-by-byte basis. Therefore, the state machine will move to IDLE state after each byte is received.
UARTCR[RDFLRFC] is programmed for a certain number of bytes received, but the actual number of bytes received is smaller.	The reception will hang. In this case, the software must monitor the UARTSR[TO] field, and move to IDLE state by setting LINCR1[SLEEP].
A STOP request arrives before the reception is completed.	The request is acknowledged only after the programmed number of data bytes are received. In other words, the STOP request is not serviced immediately. In this case, the software must monitor the UARTSR[TO] field and move the state machine to IDLE state as appropriate. The stop request will be serviced only after this is complete.
A parity error occurs during the reception of a byte.	The corresponding UARTSR[PE*n*] field is set. No interrupt is generated.
A framing error occurs during the reception of a byte.	– UARTSR[FEF] is set. – If LINIER[FEIE] = 1, an interrupt is generated. This interrupt is helpful in identifying which byte has the framing error, since there is only one register bit for framing errors.
A new byte has been received, but the last received frame has not been read from the buffer (UARTSR[RMB] has not yet been cleared by the software)	– An overrun error will occur (UARTSR[BOF] will be set). – One message will be lost (depending on the setting of LINCR1[RBLM]). – An interrupt is generated if LINIER[BOIE] is set.

26.10 Memory map and register description

The memory maps for the LINFlexD modules on this microcontroller differ by module:

- The memory map for LINFlexD_0 is shown in *Table 255*.
- The memory map for LINFlexD_1–9 is shown in *Table 256*.

See the microcontroller memory map for the base addresses.

Note: In Master mode, the registers IFCR0 - IFCR15 are not present. In Master mode, read access to IFMI register and read/write access to IFER and IFMR registers would result in transfer error being flagged.

Table 255. LINFlexD_0 memory map

Address offset	Register description	Location
0x00	LIN control register 1 (LINCR1)	*on page 631*
0x04	LIN interrupt enable register (LINIER)	*on page 634*
0x08	LIN status register (LINSR)	*on page 635*
0x0C	LIN error status register (LINESR)	*on page 638*
0x10	UART mode control register (UARTCR)	*on page 639*
0x14	UART mode status register (UARTSR)	*on page 642*
0x18	LIN timeout control status register (LINTCSR)	*on page 644*
0x1C	LIN output compare register (LINOCR)	*on page 645*
0x20	LIN timeout control register (LINTOCR)	*on page 645*
0x24	LIN fractional baud rate register (LINFBRR)	*on page 646*
0x28	LIN integer baud rate register (LINIBRR)	*on page 646*
0x2C	LIN checksum field register (LINCFR)	*on page 647*
0x30	LIN control register 2 (LINCR2)	*on page 648*
0x34	Buffer identifier register (BIDR)	*on page 649*
0x38	Buffer data register least significant (BDRL)	*on page 650*
0x3C	Buffer data register most significant (BDRM)	*on page 651*
0x40	Identifier filter enable register (IFER)	*on page 651*
0x44	Identifier filter match index (IFMI)	*on page 652*
0x48	Identifier filter mode register (IFMR)	*on page 653*
0x4C–0x88	Identifier filter control registers 0–15 (IFCR0–IFCR15)	*on page 654*
0x8C	Global control register (GCR)	*on page 655*
0x90	UART preset timeout register (UARTPTO)	*on page 656*
0x94	UART current timeout register (UARTCTO)	*on page 656*
0x98	DMA Tx enable register (DMATXE)	*on page 657*
0x9C	DMA Rx enable register (DMARXE)	*on page 658*

Table 256. LINFlexD_1–9 memory map

Address offset	Register description	Location
0x00	LIN control register 1 (LINCR1)	*on page 631*
0x04	LIN interrupt enable register (LINIER)	*on page 634*
0x08	LIN status register (LINSR)	*on page 635*
0x0C	LIN error status register (LINESR)	*on page 638*
0x10	UART mode control register (UARTCR)	*on page 639*
0x14	UART mode status register (UARTSR)	*on page 642*
0x18	LIN timeout control status register (LINTCSR)	*on page 644*
0x1C	LIN output compare register (LINOCR)	*on page 645*
0x20	LIN timeout control register (LINTOCR)	*on page 645*
0x24	LIN fractional baud rate register (LINFBRR)	*on page 646*
0x28	LIN integer baud rate register (LINIBRR)	*on page 646*
0x2C	LIN checksum field register (LINCFR)	*on page 647*
0x30	LIN control register 2 (LINCR2)	*on page 648*
0x34	Buffer identifier register (BIDR)	*on page 649*
0x38	Buffer data register least significant (BDRL)	*on page 650*
0x3C	Buffer data register most significant (BDRM)	*on page 651*
0x40	Identifier filter enable register (IFER)	*on page 651*
0x44	Identifier filter match index (IFMI)	*on page 652*
0x48	Identifier filter mode register (IFMR)	*on page 653*
0x4C	Global control register (GCR)	*on page 655*
0x50	UART preset timeout register (UARTPTO)	*on page 656*
0x54	UART current timeout register (UARTCTO)	*on page 656*
0x58	DMA Tx enable register (DMATXE)	*on page 657*
0x5C	DMA Rx enable register (DMARXE)	*on page 658*

26.10.1 LIN control register 1 (LINCR1)

Offset:0x00 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CCD(1)	CFD(1)	LASE(1)	AWUM(1)	MBL(1)				BF(1)	SFTM(1)	LBKM(1)	MME(1)	SBDT(1)	RBLM(1)	SLEEP	INIT
Reset	0	0	0	0	0	0	0	0	1	0	0	0/1(2)	0	0	1	0

1 These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

2 Resets to 0 in Slave mode and to 1 in Master mode

Figure 289. LIN control register 1 (LINCR1)

Table 257. LINCR1 field descriptions

Field	Description
CCD	Checksum Calculation disable This bit is used to disable the checksum calculation (see *Table 258*). 0: Checksum calculation is done by hardware. When this bit is reset the LINCFR register is read-only. 1: Checksum calculation is disabled. When this bit is set the LINCFR register is read/write. User can program this register to send a software calculated CRC (provided CFD is reset). **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 is used to disable the checksum field transmission (see *Table 258*). 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 LINFlexD hardware during Sleep mode. 0: The Sleep mode is exited on software request by clearing SLEEP bit. 1: The Sleep mode is exited automatically by hardware on RX dominant state detection. SLEEP bit is cleared by hardware whenever LINSR[WUF] bit 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 These bits indicate the Break length in Master mode (see *Table 259*). **Note:** These bits can be written in Initialization mode only. They are read-only in Normal or Sleep mode.

Table 257. LINCR1 field descriptions (continued)

Field	Description
BF	Bypass filter 0: No interrupt if ID does not match any filter 1: An RX interrupt is generated on ID 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 please refer to *Section 26.8.2: 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 please refer to *Section 26.8.1: Loop Back mode*. 0: Loop Back mode disable 1: Loop Back mode enable **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode
MME	Master Mode Enable 0: Slave mode enable 1: Master mode enable **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SBDT	Slave Mode Break Detection Threshold 0: 11-bit break 1: 10-bit break **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
RBLM	Receive Buffer Locked Mode 0: Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one. 1: Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SLEEP	Sleep Mode Request This bit is set by software to request LINFlexD to enter Sleep mode. This bit is cleared by software to exit Sleep mode or by hardware if AWUM bit and LINSR[WUF] bit are set (see *Table 260*).
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlexD enters Normal mode when clearing the INIT bit (see *Table 260*).

Table 258. 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 259. 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
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 260. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

26.10.2 LIN interrupt enable register (LINIER)

Offset: 0x04 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									
W	SZIE	OCIE	BEIE	CEIE	HEIE			FEIE	BOIE	LSIE	WUIE	DBFIE	DBEIETOIE	DRIE	DTIE	HRIE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 290. LIN interrupt enable register (LINIER)

Table 261. LINIER field descriptions

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0: No interrupt when SZF bit in LINESR or UARTSR is set 1: Interrupt generated when SZF bit in LINESR or UARTSR is set
OCIE	Output Compare Interrupt Enable 0: No interrupt when OCF bit in LINESR or UARTSR is set 1: Interrupt generated when OCF bit in LINESR or UARTSR is set
BEIE	Bit Error Interrupt Enable 0: No interrupt when LINESR[BEF] is set 1: Interrupt generated when LINESR[BEF] is set
CEIE	Checksum Error Interrupt Enable 0: No interrupt on Checksum error 1: Interrupt generated when LINESR[CEF] is set
HEIE	Header Error Interrupt Enable 0: No interrupt on Break Delimiter error, Synch Field error, ID field error 1: Interrupt generated on Break Delimiter error, Synch Field error, ID 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.
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

Table 261. LINIER field descriptions (continued)

Field	Description
DBFIE	Data Buffer Full Interrupt Enable 0: No interrupt when buffer data register is full 1: Interrupt generated when data buffer register is full
DBEIETOIE	Data Buffer Empty Interrupt Enable / Timeout Interrupt Enable 0: No interrupt when buffer data register is empty 1: Interrupt generated when data buffer register is empty **Note:** An interrupt is generated if this bit is set and one of the following is true: LINFlexD is in LIN mode and LINSR[DBEF] is set LINFlexD is in UART mode and UARTSR[TO] is set
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(1)
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 register(1)
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 register is set

1. Do not enable this bit in UART FIFO mode as this bit has functional significance only in LIN mode or UART buffer mode.

Note: This register programs interrupts for both LIN and UART.

26.10.3 LIN status register (LINSR)

Offset: 0x08 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

Figure 291. LIN status register (LINSR)

Table 262. LINSR field descriptions

Field	Description
LINS	LIN state LIN mode states description 0000: Sleep mode LINFlexD is in Sleep mode to save power consumption. 0001: Initialization mode LINFlexD is in Initialization mode. 0010: Idle This state is entered on several events: – SLEEP bit and INIT in LINCR1 register 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. Refer to LINCR1 register 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 ID 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 register. 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
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] is reset and reception starts then this bit is set. In this case, user cannot set LINCR2[DTRQ].

Table 262. LINSR field descriptions (continued)

Field	Description
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 LINFlexD 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 LINIER[WUIE] 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 LINCR2[IOBE] is reset.
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 LINCR1[BF] is set – no match in any filter and LINCR1[BF] is set – TX filter match

26.10.4 LIN error status register (LINESR)

Offset: 0x0C 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

Figure 292. LIN error status register (LINESR)

Table 263. 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. 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, LINFlexD moves to Idle state. If LTOM bit in LINTCSR register is set then OCF is reset by hardware in Initialization mode. If LTOM bit is reset, 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 LINFlexD 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.
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 register 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.

Table 263. LINESR field descriptions (continued)

Field	Description
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlexD 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.

26.10.5 UART mode control register (UARTCR)

Offset: 0x10 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	TDFLTFC(1)			RDFLRFC(1)			RFBM	TFBM(2)	WL[1](2)	PC1(2)	RXEN	TXEN	PC0(2)	PCE(2)	WL[0](2)	UART(2)
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These fields are read/write in UART buffer mode and read-only in other modes.

2 These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

Figure 293. UART mode control register (UARTCR)

Table 264. UARTCR field descriptions

Field	Description
TDFLTFC	Transmitter data field length / Tx FIFO counter This field has one of two functions depending on the mode of operation as follows: – When LINFlexD is in UART buffer mode (TFBM = 0), TDFLTFC defines the number of bytes to be transmitted. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit): 0bX00: 1 byte 0bX01: 2 bytes 0bX10: 3 bytes 0bX11: 4 bytes When the UART data length is configured as half-word (WL = 0b10 or 0b11), the only valid values for TDFLTFC are 0b001 and 0b011. – When LINFlexD is in UART FIFO mode (TFBM = 1), TDFLTFC contains the number of entries (bytes) of the Tx FIFO. The field is read-only in this configuration. The permissible values are as follows: 0b000: Empty 0b001: 1 byte 0b010: 2 bytes 0b011: 3 bytes 0b100: 4 bytes All other values are reserved. This field is meaningful and can be programmed only when the UART bit is set.
RDFLRFC	Receiver data field length / Rx FIFO counter This field has one of two functions depending on the mode of operation as follows: – When LINFlexD is in UART buffer mode (RFBM = 0), RDFLRFC defines the number of bytes to be received. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit): 0bX00: 1 byte 0bX01: 2 bytes 0bX10: 3 bytes 0bX11: 4 bytes When the UART data length is configured as half-word (WL = 0b10 or 0b11), the only valid values for RDFLRFC are 0b001 and 0b011. – When LINFlexD is in UART FIFO mode (RFBM = 1), RDFLRFC contains the number of entries (bytes) of the Rx FIFO. The field is read-only in this configuration. The permissible values are as follows: 0b000: Empty 0b001: 1 byte 0b010: 2 bytes 0b011: 3 bytes 0b100: 4 bytes All other values are reserved. This field is meaningful and can be programmed only when the UART bit is set.
RFBM	Rx FIFO/buffer mode 0 Rx buffer mode enabled 1 Rx FIFO mode enabled (mandatory in DMA Rx mode) This field can be programmed in initialization mode only when the UART bit is set.

Table 264. UARTCR field descriptions (continued)

Field	Description
TFBM	Tx FIFO/buffer mode 0 Tx buffer mode enabled 1 Tx FIFO mode enabled (mandatory in DMA Tx mode) This field can be programmed in initialization mode only when the UART bit is set.
RXEN	Receiver Enable 0: Receiver disabled 1: Receiver enabled This field can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0: Transmitter disabled 1: Transmitter enabled This field 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.
PC[1:0]	Parity control 00 Parity sent is even 01 Parity sent is odd 10 A logical 0 is always transmitted/checked as parity bit 11 A logical 1 is always transmitted/checked as parity bit This field can be programmed in initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0: Parity transmit/check disabled 1: Parity transmit/check enabled This field can be programmed in Initialization mode only when the UART bit is set.
WL[1:0]	Word length in UART mode 00 7 bits data + parity 01 8 bits data when PCE = 0 or 8 bits data + parity when PCE = 1 10 15 bits data + parity 11 16 bits data when PCE = 0 or 16 bits data + parity when PCE = 1 This field can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0: LIN mode 1: UART mode This field can be programmed in Initialization mode only.

26.10.6 UART mode status register (UARTSR)

Offset: 0x14 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	TO	DRFRFE	DTFTFF	NF
W	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 294. UART mode status register (UARTSR)

Table 265. 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). 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). 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). 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). No interrupt is generated if this error occurs. 0: No parity error 1: Parity error

Table 265. 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 LINFlexD has detected a framing error (invalid stop bit).
BOF	FIFO/buffer overrun flag This bit is set by hardware when a new data byte is received and the RMB bit is not cleared in UART buffer mode. In UART FIFO mode, this bit is set when there is a new byte and the Rx FIFO is full. In UART FIFO mode, once Rx FIFO is full, the new received message is discarded regardless of the value of LINCR1[RBLM]. If LINCR1[RBLM] = 1, the new byte received is discarded. If LINCR1[RBLM] = 0, the new byte overwrites buffer. This field can be cleared by writing a 1 to it. An interrupt is generated if LINIER[BOIE] is set.
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 LINFlexD 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.
TO	Timeout The LINFlexD controller sets this field when a UART timeout occurs — that is, when the value of UARTCTO becomes equal to the preset value of the timeout (UARTPTO register setting). This field should be cleared by software. The GCR[SR] field should be used to reset the receiver FSM to idle state in case of UART timeout for UART reception depending on the application both in buffer and FIFO mode. An interrupt is generated when LINIER[DBEIETOIE] is set on the Error interrupt line in UART mode.
DRFRFE	Data reception completed flag / Rx FIFO empty flag The LINFlexD controller sets this field as follows: – In UART buffer mode (RFBM = 0), it indicates that the number of bytes programmed in RDFL has been received. This field should be cleared by software. An interrupt is generated if LINIER[DRIE] is set. This field is set in case of framing error, parity error, or overrun. This field reflects the same value as in LINESR when in Initialization mode and UART bit is set. – In UART FIFO mode (RFBM = 1), it indicates that the Rx FIFO is empty. This field is a read-only field used internally by the DMA Rx interface.
DTFTFF	Data transmission completed flag / Tx FIFO full flag The LINFlexD controller sets this field as follows: – In UART buffer mode (TFBM = 0), it indicates that the data transmission is completed. This field should be cleared by software. An interrupt is generated if LINIER[DTIE] is set. This field reflects the same value as in LINESR when in Initialization mode and UART bit is set. – In UART FIFO mode (TFBM = 1), it indicates that the Tx FIFO is full. This field is a read-only field used internally by the DMA Tx interface.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

26.10.7 LIN timeout control status register (LINTCSR)

Offset: 0x18 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(1)	IOT(1)	TOCE	CNT							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

1 These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

Figure 295. LIN timeout control status register (LINTCSR)

Table 266. LINTCSR field descriptions

Name	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 does not reset to Idle on timeout 1: LIN state machine resets to Idle on timeout event This feature is applicable only when MODE bit in LINTCSR is cleared.
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 ans software cannot modify it.
CNT	Counter Value These bits indicate the LIN Timeout counter value.

26.10.8 LIN output compare register (LINOCR)

Offset: 0x1C Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OC2[1]								OC1[1]							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1 If LINTCSR[LTOM] = 0, these fields are read-only.(These fields are writable only in Output Compare mode)

Figure 296. LIN output compare register (LINOCR)

Table 267. LINOCR field descriptions

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of LINTCSR[CNT].
OC1	Output compare 1 value These bits contain the value to be compared to the value of LINTCSR[CNT].

26.10.9 LIN timeout control register (LINTOCR)

Offset: 0x20 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[3]						
W																
Reset	0	0	0	0	1	1	1	0	0	0	0/1[1]	0/1[2]	1	1	0	0

1 Resets to 1 in Slave mode and to 0 in Master mode
2 Resets to 0 in Slave mode and to 1 in Master mode
3 HTO field can only be written in slave mode, LINCR1[MME] = 0.

Figure 297. LIN timeout control register (LINTOCR)

Table 268. LINTOCR field descriptions

Field	Description
RTO	Response timeout value This register 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 register contains the header timeout duration (in bit time). This value does not include the first 11 dominant bits of the Break. The reset value depends on which mode LINFlexD is in. HTO can be written only for Slave mode.

26.10.10 LIN fractional baud rate register (LINFBRR)

Offset: 0x24 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(1)			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 This field is writable only in Initialization mode, LINCR1[INIT] = 1.

Figure 298. LIN timeout control register (LINTOCR)

Table 269. LINFBRR field descriptions

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlexD divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16. This register can be written in Initialization mode only, LINCR1[INIT] = 1.

26.10.11 LIN integer baud rate register (LINIBRR)

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	DIV_M(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 This field is writable only in Initialization mode (LINCR1[INIT] = 1).

Figure 299. LIN integer baud rate register (LINIBRR)

Table 270. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa These bits define the LINFlexD divider (LFDIV) mantissa value (see *Table 271*). This register can be written in Initialization mode only.

Table 271. Integer baud rate selection

DIV_M	Mantissa
0x0	LIN clock disabled
0x1	1
...	...
0xFFFFE	1048574
0xFFFFF	1048575

26.10.12 LIN checksum field register (LINCFR)

Offset: 0x2C 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

Figure 300. LIN checksum field register (LINCFR)

Table 272. LINCFR field descriptions

Field	Description
CF	Checksum bits When LINCR1[CCD] is cleared, these bits are read-only. When LINCR1[CCD] is set, these bits are read/write. See *Table 258*.

26.10.13 LIN control register 2 (LINCR2)

Offset: 0x30 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(1)	IOPE(1)	WURQ(2)	DDRQ(3)	DTRQ(3)	ABRQ(3)	HTRQ(3)	0	0	0	0	0	0	0	0
W																
Reset	0	1	0/1(2)	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 301. LIN control register 2 (LINCR2)

1. These fields are writable only in Initialization mode (LINCR1[INIT] = 1.
2. Resets to 1 in Slave mode and to 0 in Master mode.
3. These fields are cleared by hardware.

Table 273. 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 (LINCR1[INIT]) = 1.
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 (LINCR1[INIT]) = 1.
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 LINFlexD 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 DIR bit in BIDR is set and header transmission is completed.

Table 273. LINCR2 field descriptions (continued)

Field	Description
ABRQ	Abort Request Set by software to abort the current transmission. Cleared by hardware when the transmission has been aborted. LINFlexD 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.

26.10.14 Buffer identifier register (BIDR)

This register contains the fields that identify a transaction and provide other information related to it.

All the fields in this register must be updated when an ID filter (enabled) in slave mode (Tx or Rx) matches the ID received.

Offset: 0x34 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

Figure 302. Buffer identifier register (BIDR)

Table 274. BIDR field descriptions

Field	Description
DFL	Data Field Length These bits define the number of data bytes in the response part of the frame. DFL = Number of data bytes - 1. Normally, LIN uses only DFL[0:2] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[0:2] and DFL[0:5]. DFL[3:5] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0: LINFlexD receives the data and copy them in the BDR registers. 1: LINFlexD transmits the data from the BDR registers.

Table 274. BIDR field descriptions (continued)

Field	Description
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 below.
ID	Identifier Identifier part of the identifier field without the identifier parity.

26.10.15 Buffer data register least significant (BDRL)

Offset: 0x38 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

Figure 303. Buffer data register least significant (BDRL)

Table 275. 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

26.10.16 Buffer data register most significant (BDRM)

Offset: 0x3C 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

Figure 304. Buffer data register most significant (BDRM)

Table 276. 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

26.10.17 Identifier filter enable register (IFER)

Offset: 0x40 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	FACT(1)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 305. Identifier filter enable register (IFER)

1. This field is writable only in Initialization mode (LINCR1[INIT] = 1).

Table 277. IFER field descriptions

Field	Description
FACT	Filter activation The software sets the bit FACT[x] to activate the filters x in identifier list mode. In identifier mask mode bits FACT(2n + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2n. 0 Filter x is deactivated 1 Filter x is activated

26.10.18 Identifier filter match index (IFMI)

Offset: 0x44 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	IFMI				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 306. Identifier filter match index (IFMI)

Table 278. IFMI field descriptions

Field	Description
IFMI	Filter match index This register contains the index corresponding to the received ID. It can be used to directly write or read the data in RAM (refer to *Section 26.7.2: Slave mode*, for more details). When no filter matches, IFMI = 0. When Filter n is matching, IFMI = n + 1.

26.10.19 Identifier filter mode register (IFMR)

Offset:0x48 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

Figure 307. Identifier filter mode register (IFMR)

Table 279. IFMR field descriptions

Field	Description
IFM	Filter mode 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*). (See *Table 280*.)

Table 280. IFMR[IFM] configuration

Bit	Value	Result
IFM[0]	0	Filters 0 and 1 are in identifier list mode.
	1	Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).
IFM[1]	0	Filters 2 and 3 are in identifier list mode.
	1	Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).
IFM[2]	0	Filters 4 and 5 are in identifier list mode.
	1	Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).
IFM[3]	0	Filters 6 and 7 are in identifier list mode.
	1	Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).
IFM[4]	0	Filters 8 and 9 are in identifier list mode.
	1	Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).
IFM[5]	0	Filters 10 and 11 are in identifier list mode.
	1	Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10).
IFM[6]	0	Filters 12 and 13 are in identifier list mode.
	1	Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12).
IFM[7]	0	Filters 14 and 15 are in identifier list mode.
	1	Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14).

26.10.20 Identifier filter control registers (IFCR0–IFCR15)

The function of these registers is different depending on which mode the LINFlexD controller is in, as described in *Table 281*.

Table 281. IFCR functionality based on mode

Mode	IFCR functionality
Identifier list	Each IFCR register acts as a filter.
Identifier mask	If a = (number of filters) / 2, and n = 0 to (a - 1), then IFCR[2n] acts as a filter and IFCR[2n+1] acts as the mask for IFCR[2n].

Note: These registers are available on LINFlexD_0 only.

Offsets: 0x4C–0x88 (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	DFL(1)						DIR(1)	CCS(1)	0	0	ID(1)					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

Figure 308. Identifier filter control registers (IFCR0–IFCR15)

Table 282. IFCR 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: LINFlexD receives the data and copy them in the BDRL and BDRM registers. 1: LINFlexD 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 below.
ID	Identifier Identifier part of the identifier field without the identifier parity.

26.10.21 Global control register (GCR)

This register can be programmed only in Initialization mode. The configuration specified in this register applies in both LIN and UART modes.

Offset: 0x8C (for LINFlexD_0 only), 0x4C (for LINFlexD_1–9) 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	TDFBM(1)	RDFBM(1)	TDLIS(1)	RDLIS(1)	STOP(1)	0
W																SR(1)
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 This field is writable only in Initialization mode (LINCR1[INIT] = 1).

Figure 309. Global control register (GCR)

Table 283. GCR field descriptions

Field	Description
TDFBM	Transmit data first bit MSB This field controls the first bit of transmitted data (payload only) as MSB/LSB in both UART and LIN modes. 0 The first bit of transmitted data is LSB – that is, the first bit transmitted is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)). 1 The first bit of transmitted data is MSB – that is, the first bit transmitted is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).
RDFBM	Received data first bit MSB This field controls the first bit of received data (payload only) as MSB/LSB in both UART and LIN modes. 0 The first bit of received data is LSB – that is, the first bit received is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)). 1 The first bit of received data is MSB – that is, the first bit received is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).
TDLIS	Transmit data level inversion selection This field controls the data inversion of transmitted data (payload only) in both UART and LIN modes. 0 Transmitted data is not inverted. 1 Transmitted data is inverted.
RDLIS	Received data level inversion selection This field controls the data inversion of received data (payload only) in both UART and LIN modes. 0 Received data is not inverted. 1 Received data is inverted.

Table 283. GCR field descriptions (continued)

Field	Description
STOP	Stop bit configuration This field controls the number of stop bits in transmitted data in both UART and LIN modes. The stop bit is configured for all the fields (delimiter, sync, ID, checksum, and payload). 0 One stop bit 1 Two stop bits
SR	Soft reset If the software writes a “1” to this field, the LINFlexD controller executes a soft reset in which the FSMs, FIFO pointers, counters, timers, status registers, and error registers are reset but the configuration registers are unaffected. This field should be cleared by software to perform further operations (the field is not cleared by hardware). This field always reads “0”.

26.10.22 UART preset timeout register (UARTPTO)

This register contains the preset timeout value in UART mode, and is used to monitor the IDLE state of the reception line. The timeout detection uses this register and the UARTCTO register described in *Section 26.10.23: UART current timeout register (UARTCTO)*.

Offset: 0x90 (for LINFlexD_0 only), 0x50 (for LINFlexD_1–9) 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	PTO											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 310. UART preset timeout register (UARTPTO)

Table 284. UARTPTO field descriptions

Field	Description
PTO	Preset value of the timeout counter Do not set PTO = 0 (otherwise, UARTSR[TO] would immediately be set).

26.10.23 UART current timeout register (UARTCTO)

This register contains the current timeout value in UART mode, and is used in conjunction with the UARTPTO register (see *Section 26.10.22: UART preset timeout register (UARTPTO)*) to monitor the IDLE state of the reception line. UART timeout works in both CPU and DMA modes.

The timeout counter:

- Starts at zero and counts upward
- Is clocked with the baud rate clock prescaled by a hard-wired scaling factor of 16
- Is automatically enabled when UARTCR[RXEN] = 1

Offset: 0x94 (for LINFlexD_0 only), 0x54 (for LINFlexD_1–9) 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	CTO											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 311. UART current timeout register (UARTCTO)

Table 285. UARTCTO field descriptions

Field	Description
CTO	Current value of the timeout counter This field is reset whenever one of the following occurs: – A new value is written to the UARTPTO register – The value of this field matches the value of UARTPTO[PTO] – A hard or soft reset occurs – New incoming data is received When CTO matches the value of UARTPTO[PTO], UARTSR[TO] is set.

26.10.24 DMA Tx enable register (DMATXE)

This register enables the DMA Tx interface.

Offset: 0x98 (for LINFlexD_0 only), 0x58 (for LINFlexD_1–9) 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	DTE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 312. DMA Tx enable register (DMATXE)

Table 286. DMATXE field descriptions

Field	Description
DTE*n*	DMA Tx channel *n* enable 0 DMA Tx channel *n* disabled 1 DMA Tx channel *n* enabled **Note:** When DMATXE = 0x0, the DMA Tx interface FSM is forced (soft reset) into the IDLE state.

26.10.25 DMA Rx enable register (DMARXE)

This register enables the DMA Rx interface.

Offset: 0x9C (for LINFlexD_0 only), 0x5C (for LINFlexD_1–9) 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	DRE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 313. DMA Rx enable register (DMARXE)

Table 287. DMARXE field descriptions

Field	Description
DRE*n*	DMA Rx channel *n* enable 0 DMA Rx channel *n* disabled 1 DMA Rx channel *n* enabled **Note:** When DMARXE = 0x0, the DMA Rx interface FSM is forced (soft reset) into the IDLE state.

26.11 DMA interface

The LINFlexD DMA interface offers a parametric and programmable solution with the following features:

- LIN Master node, TX mode: single DMA channel
- LIN Master node, RX mode: single DMA channel
- LIN Slave node, TX mode: 1 to N DMA channels where N = max number of ID filters
- LIN Slave node, RX mode: 1 to N DMA channels where N = max number of ID filters
- UART node, TX mode: single DMA channel
- UART node, RX mode: single DMA channel + timeout

The LINFlexD controller interacts with an enhanced direct memory access (eDMA) controller; see the description of that controller for details on its operation and the transfer control descriptors (TCDs) referenced in this section.

26.11.1 Master node, TX mode

On a master node in TX mode, the DMA interface requires a single TX channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in *Figure 314*.

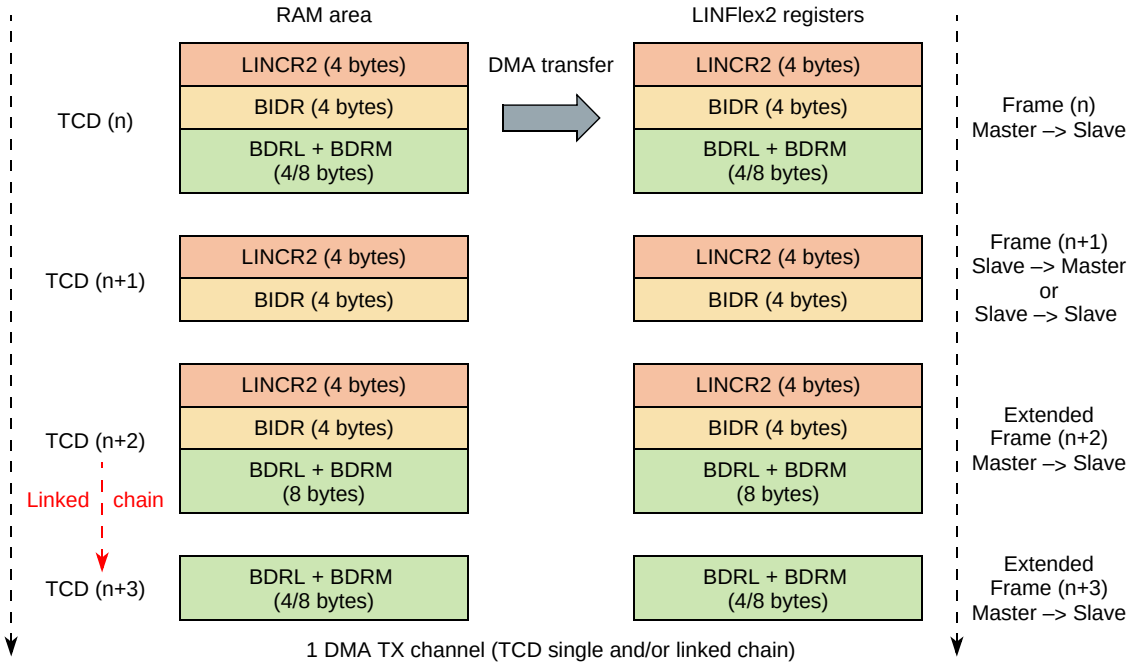


Figure 314. TCD chain memory map (master node, TX mode)

The TCD chain of the DMA Tx channel on a master node supports:

- Master to Slave: transmission of the entire frame (header + data)
- Slave to Master: transmission of the header. The data reception is controlled by the Rx channel on the master node.
- Slave to Slave: transmission of the header.

The register settings for the LINCR2 and BIDR registers for each class of LIN frame are shown in *Table 288*.

Table 288. Register settings (master node, TX mode)

LIN frame	LINCR2	BIDR
Master to Slave	DDRQ=1 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 1 (TX)
Slave to Master	DDRQ=0 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 0 (RX)
Slave to Slave	DDRQ=1 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 0 (RX)

The concept FSM to control the DMA TX interface is shown in *Figure 315*. The DMA TX FSM will move to IDLE state immediately at next clock edge if DMATXE[0] = 0.

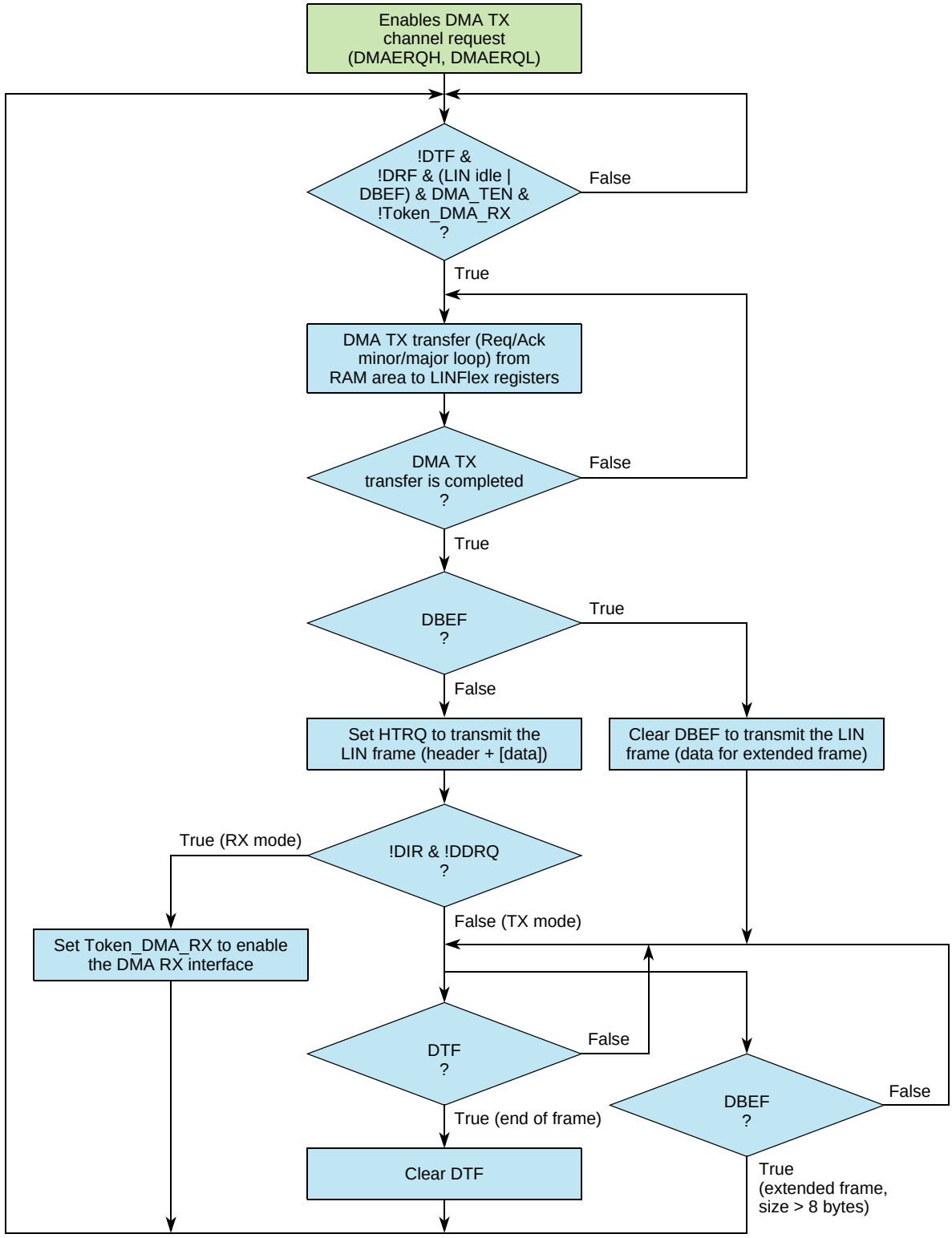


Figure 315. FSM to control the DMA TX interface (master node)

The TCD settings (word transfer) are shown in *Table 289*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfers are allowed.

Table 289. TCD settings (master node, TX mode)

TCD field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	[4 + 4] + 0/4/8 = N	Data buffer is stuffed with dummy bytes if the length is not word aligned. LINCR2 + BIDR + BDRL + BDRM
SADDR[31:0]	RAM address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	LINCR2 address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

26.11.2 Master node, RX mode

On a master node in RX mode, the DMA interface requires a single RX channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in *Figure 316*.

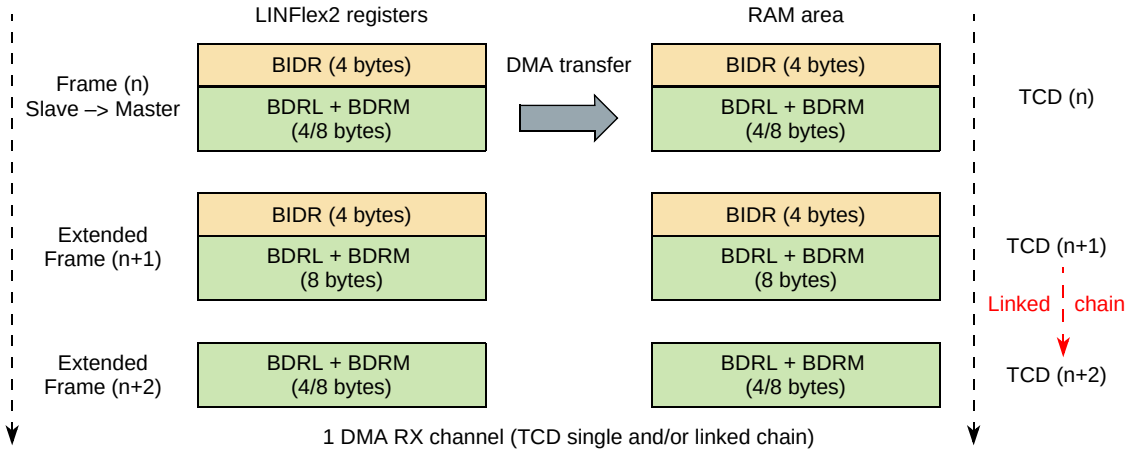


Figure 316. TCD chain memory map (master node, RX mode)

The TCD chain of the DMA Rx channel on a master node supports Slave-to-Master reception of the data field.

The BIDR register is optionally copied into the RAM area. This BIDR field (part of FIFO data) contains the ID of each message to allow the CPU to figure out which ID was received by the LINFlexD DMA if only the “one DMA channel” setup is used.

The concept FSM to control the DMA RX interface is shown in *Figure 317*. The DMA RX FSM will move to IDLE state immediately at next clock edge if DMARXE[0]=0.

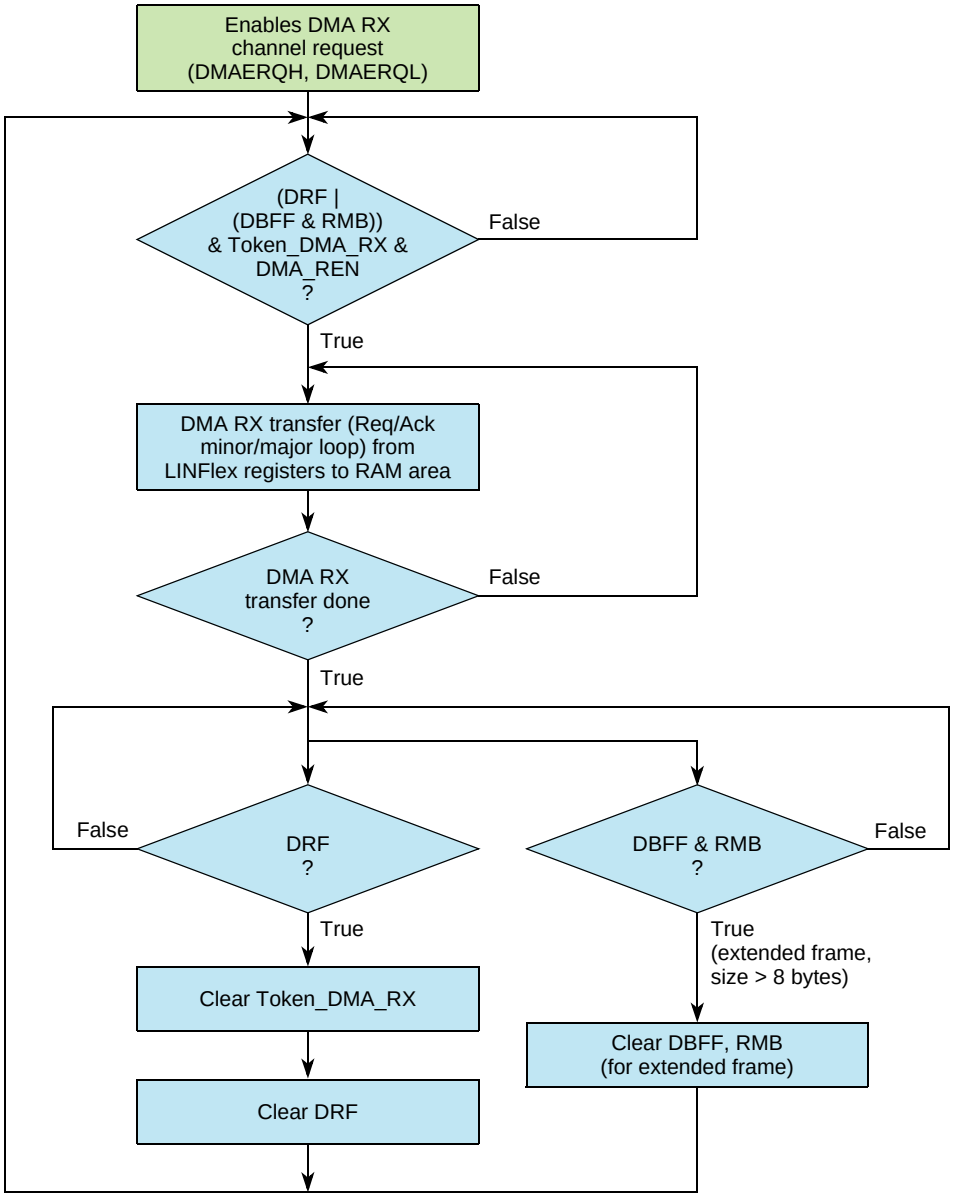


Figure 317. FSM to control the DMA RX interface (master node)

The TCD settings (word transfer) are shown in *Table 290*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

Table 290. TCD settings (master node, RX mode)

TCD field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	[4] + 4/8 = N	Data buffer is stuffed with dummy bytes if the length is not word aligned. BIDR + BDRL + BDRM
SADDR[31:0]	BIDR address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	RAM address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

26.11.3 Slave node, TX mode

On a slave node in TX mode, the DMA interface requires a DMA TX channel for each ID filter programmed in TX mode. In case a single DMA TX channel is available, a single ID field filter must be programmed in TX mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in *Figure 318*.

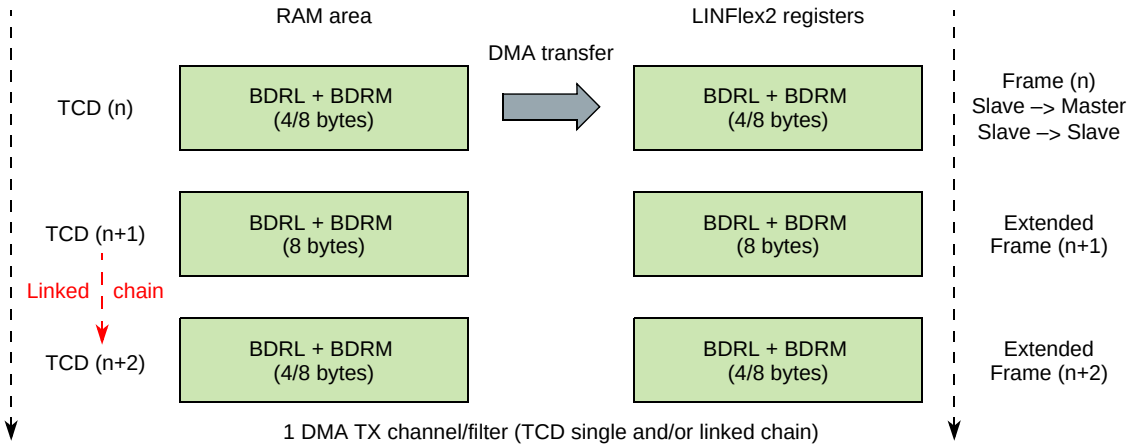


Figure 318. TCD chain memory map (slave node, TX mode)

The TCD chain of the DMA Tx channel on a slave node supports:

- Slave to Master: transmission of the data field
- Slave to Slave: transmission of the data field

The register settings of the LINCR2, IFER, IFMR, and IFCR registers are shown in *Table 291*.

Table 291. Register settings (slave node, TX mode)

LIN frame	LINCR2	IFER	IFMR	IFCR
Slave to Master or Slave to Slave	DDRQ = 0 DTRQ = 0 HTRQ = 0	To enable an ID filter (Tx mode) for each DMA TX channel	- Identifier list mode - Identifier mask mode	DFL = payload size ID = address CCS = checksum DIR = 1(TX)

The concept FSM to control the DMA Tx interface is shown in *Figure 319*. DMA TX FSM will move to idle state if DMATXE[*x*] = 0, where *x* = IFMI – 1.

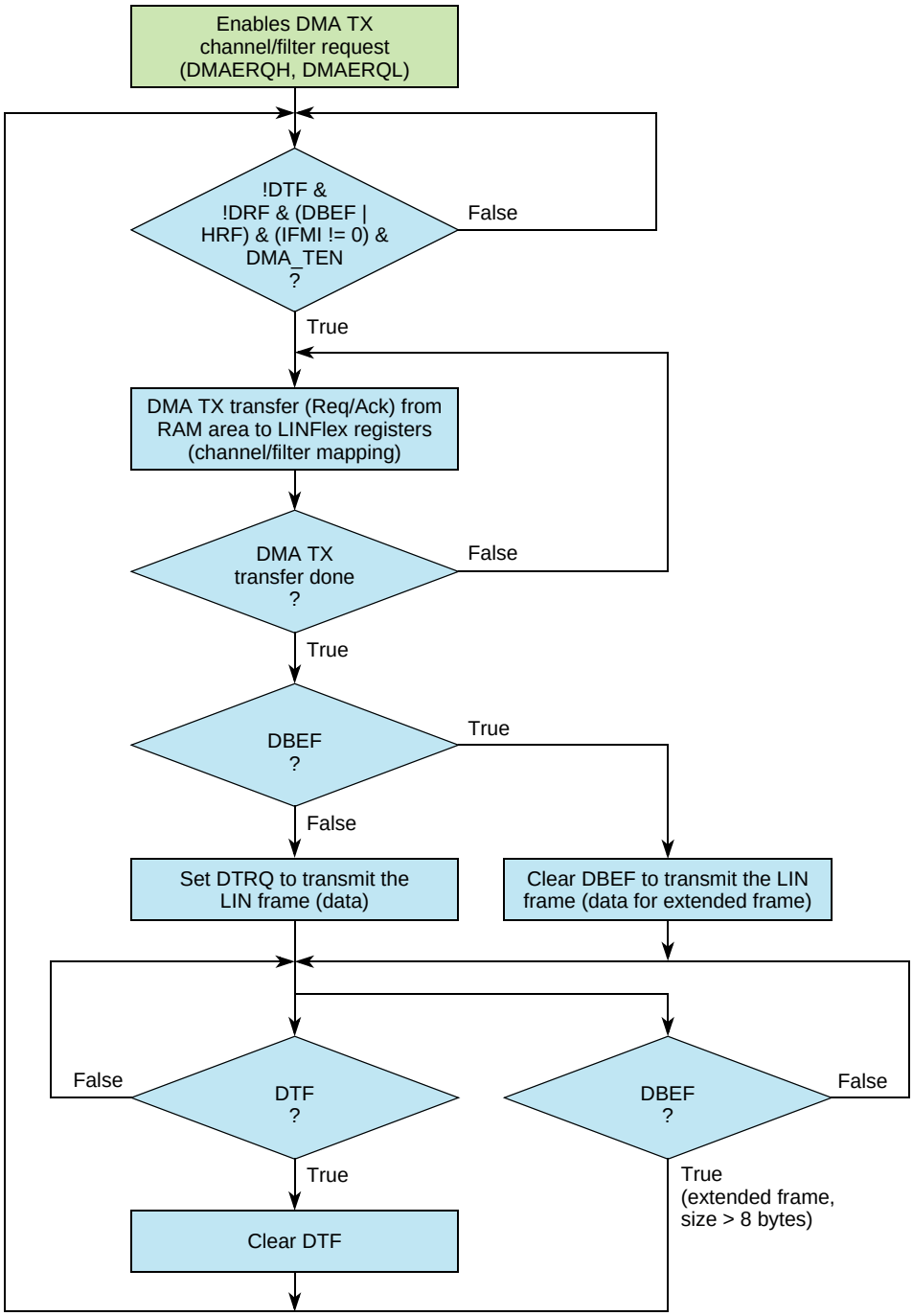


Figure 319. FSM to control the DMA TX interface (slave node)

The TCD settings (word transfer) are shown in *Table 292*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

Table 292. TCD settings (slave node, TX mode)

TCD field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	4/8 = N	Data buffer is stuffed with dummy bytes if the length is not word aligned. BDRL + BDRM
SADDR[31:0]	RAM address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	BDRL address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

26.11.4 Slave node, RX mode

On a slave node in RX mode, the DMA interface requires a DMA RX channel for each ID filter programmed in RX mode. In case a single DMA RX channel is available, a single ID field filter must be programmed in RX mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in *Figure 320*.

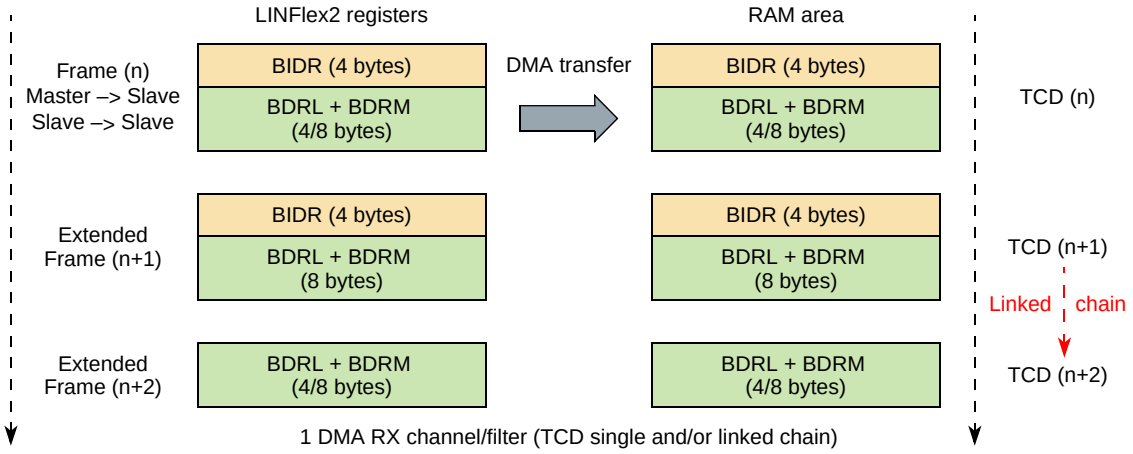


Figure 320. TCD chain memory map (slave node, RX mode)

The TCD chain of the DMA RX channel on a slave node supports:

- Master to Slave: reception of the data field.
- Slave to Slave: reception of the data field.

The register setting of the LINCR2, IFER, IFMR, and IFCR registers are given in *Table 293*.

Table 293. Register settings (slave node, RX mode)

LIN frame	LINCR2	IFER	IFMR	IFCR
Master to Slave or Slave to Slave	DDRQ = 0 DTRQ = 0 HTRQ = 0	To enable an ID filter (Rx mode) for each DMA RX channel	- Identifier list mode - Identifier mask mode	DFL = payload size ID = address CCS = checksum DIR = 0 (RX)

The concept FSM to control the DMA Rx interface is shown in *Figure 321*. DMA RX FSM will move to idle state if DMARXE[*x*] = 0 where *x* = IFMI - 1.

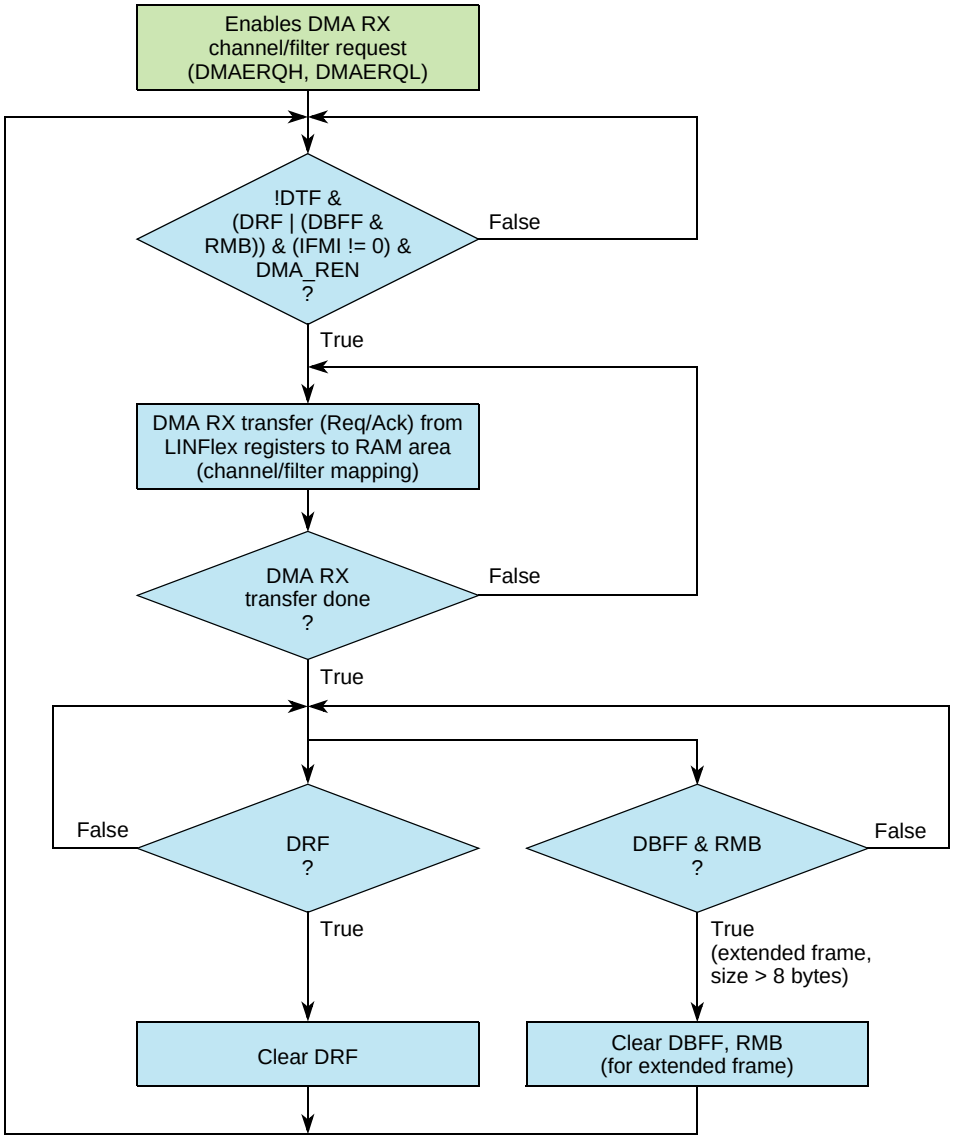


Figure 321. FSM to control the DMA RX interface (slave node)

The TCD settings (word transfer) are shown in *Table 294*. All other TCD fields = 0. TCD settings based on half-word or byte transfer are allowed.

Table 294. TCD settings (slave node, RX mode)

TCD Field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	[4] + 4/8 = N	Data buffer is stuffed with dummy bytes if the length is not word aligned. BIDR + BDRL + BDRM
SADDR[31:0]	BDRL address	

Table 294. TCD settings (slave node, RX mode) (continued)

TCD Field	Value	Description
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	RAM address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

26.11.5 UART node, TX mode

In UART TX mode, the DMA interface requires a DMA TX channel. A single TCD can control the transmission of an entire Tx buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in *Figure 322*.

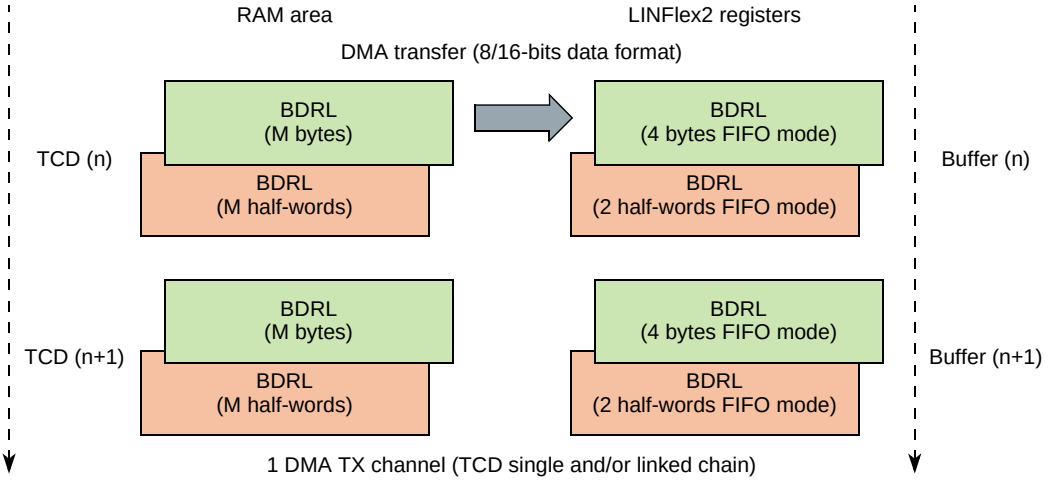


Figure 322. TCD chain memory map (UART node, TX mode)

The UART TX buffer must be configured in FIFO mode in order to:

- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from the RAM to the FIFO
- Use low priority DMA channels
- Support the UART baud rate (2 Mb/s) without underrun events

The Tx FIFO size is:

- 4 bytes in 8-bit data format
- 2 half-words in 16-bit data format

A DMA request is triggered by FIFO not full (TX) status signals.

The concept FSM to control the DMA TX interface is shown in *Figure 323*. DMA TX FSM will move to idle state if DMATXE[0] = 0.

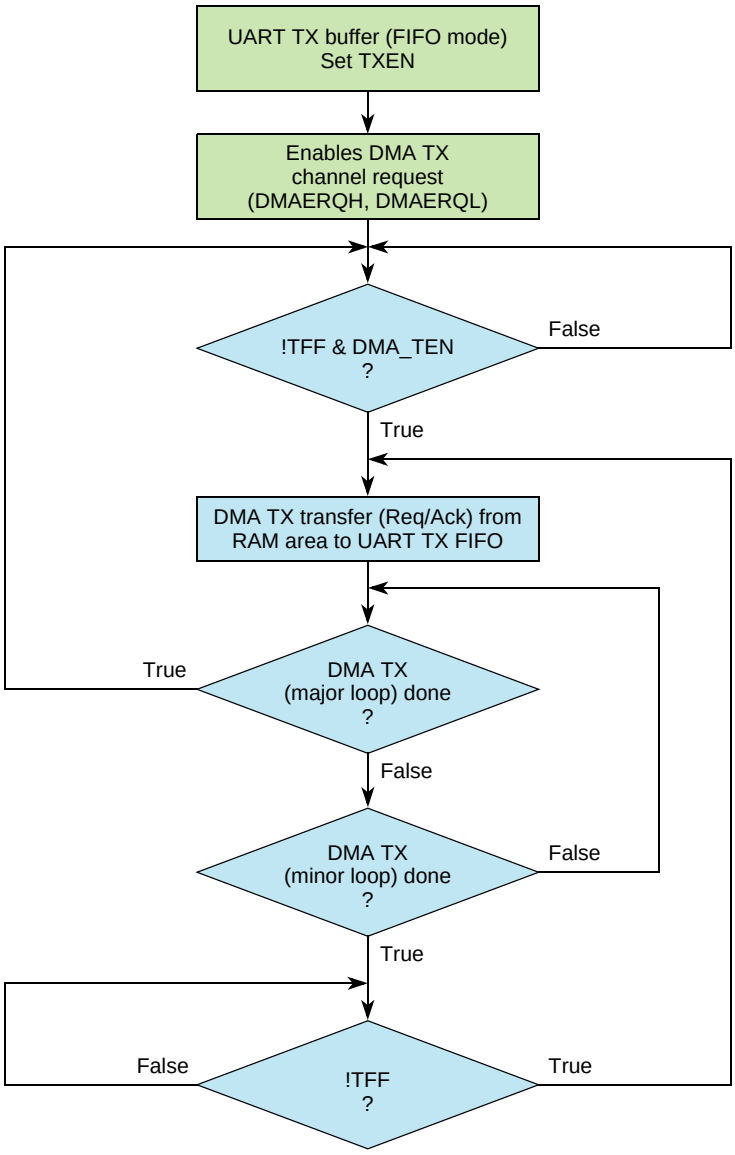


Figure 323. FSM to control the DMA TX interface (UART node)

The TCD settings (typical case) are shown in *Table 295*. All other TCD fields = 0. The minor loop transfers a single byte/half-word as soon a free entry is available in the Tx FIFO.

Table 295. TCD settings (UART node, TX mode)

TCD Field	Value		Description
	8-bit data	16-bit data	
CITER[14:0]	M		Multiple iterations for the “major” loop
BITER[14:0]	M		Multiple iterations for the “major” loop
NBYTES[31:0]	1	2	Minor loop transfer = 1 or 2 bytes
SADDR[31:0]	RAM address		
SOFF[15:0]	1	2	Byte/Half-word increment
SSIZE[2:0]	0	1	Byte/Half-word transfer
SLAST[31:0]	-M	-M * 2	
DADDR[31:0]	BDRL address		DADDR = BDRL + 0x3 for byte transfer DADDR = BDRL + 0x2 for half-word transfer
DOFF[15:0]	0		No increment (FIFO)
DSIZE[2:0]	0	1	Byte/Half-word transfer
DLAST_SGA[31:0]	0		No scatter/gather processing
INT_MAJ	0/1		Interrupt disabled/enabled
D_REQ	1		Only on the last TCD of the chain.
START	0		No software request

26.11.6 UART node, RX mode

In UART RX mode, the DMA interface requires a DMA RX channel. A single TCD can control the reception of an entire Rx buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in *Figure 324*.

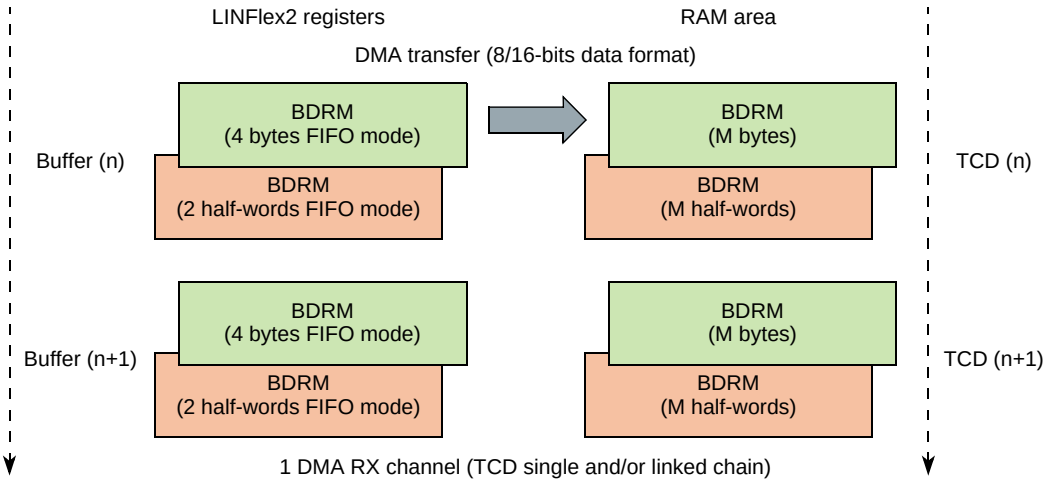


Figure 324. TCD chain memory map (UART node, RX mode)

The UART RX buffer must be configured in FIFO mode in order to:

- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from the FIFO to the RAM
- Use low priority DMA channels
- Support high UART baud rate (at least 2 Mb/s) without overrun events

The Rx FIFO size is:

- 4 bytes in 8-bit data format
- 2 half-words in 16-bit data format

This is sufficient because just one byte allows a reaction time of about 3.8 µs (at 2 Mbit/s), corresponding to about 450 clock cycles at 120 MHz, before the transmission is affected. A DMA request is triggered by FIFO not empty (RX) status signals.

The concept FSM to control the DMA Rx interface is shown in *Figure 325*. DMA Rx FSM will move to idle state if DMARXE[0] = 0.

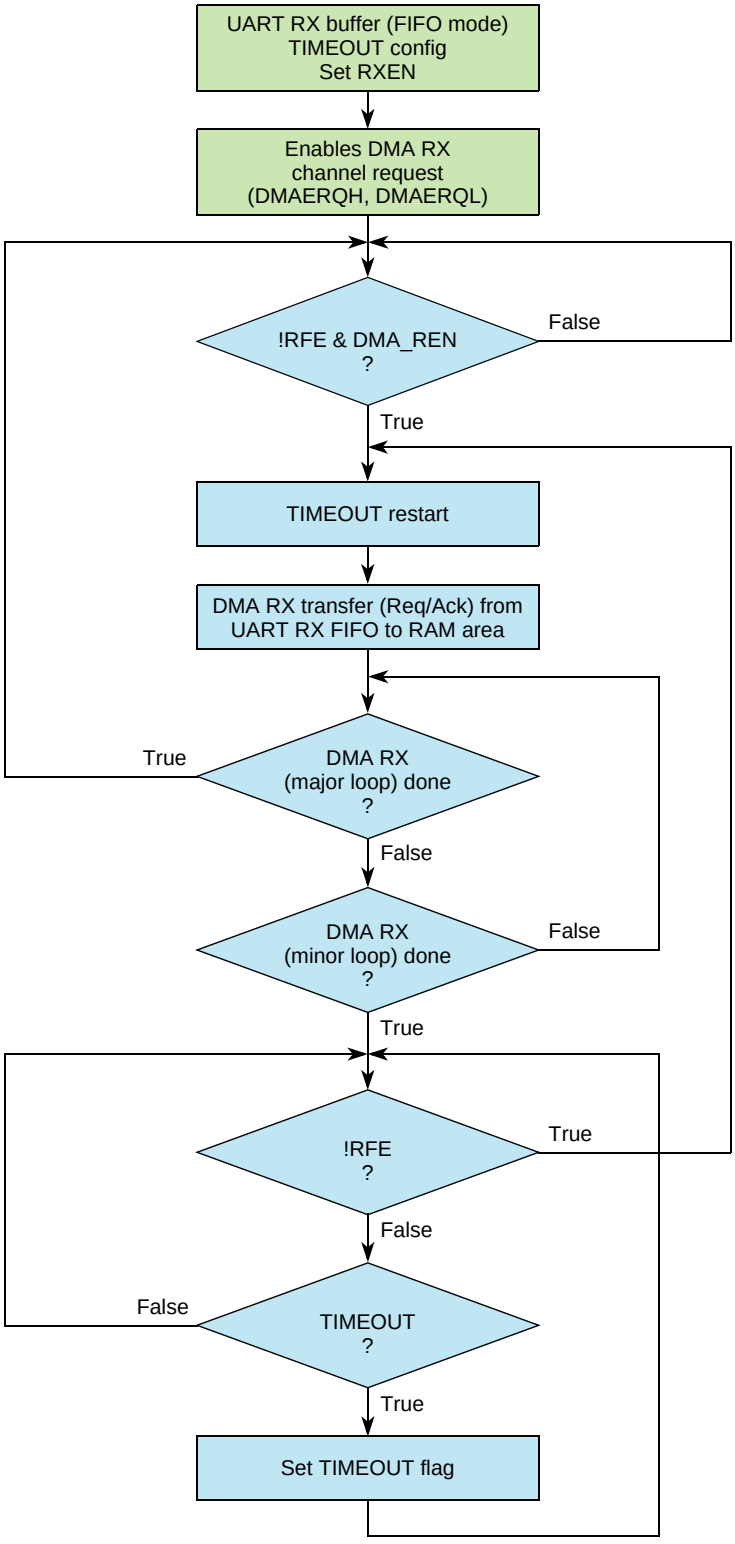


Figure 325. FSM to control the DMA RX interface (UART node)

The TCD settings (typical case) are shown in *Table 296*. All other TCD fields = 0. The minor loop transfers a single byte/half-word as soon an entry is available in the Rx FIFO. A new software reset bit is required that allows the LINFlexD FSMs to be reset in case this timeout state is reached or in any other case. Timeout counter can be re-written by software at any time to extend timeout period.

Table 296. TCD settings (UART node, RX mode)

TCD Field	Value		Description
	8 bits data	16 bits data	
CITER[14:0]	M		Multiple iterations for the “major” loop
BITER[14:0]	M		Multiple iterations for the “major” loop
NBYTES[31:0]	1	2	Minor loop transfer = 1 or 2 bytes
SADDR[31:0]	BDRM address		SADDR = BDRM + 0x3 for byte transfer SADDR = BDRM + 0x2 for half-word transfer
SOFF[15:0]	0		No increment (FIFO)
SSIZE[2:0]	0	1	Byte/Half-word transfer
SLAST[31:0]	0		
DADDR[31:0]	RAM address		
DOFF[15:0]	1	2	Byte/Half-word increment
DSIZE[2:0]	0	1	Byte/Half-word transfer
DLAST_SGA[31:0]	-M	-M * 2	No scatter/gather processing
INT_MAJ	0/1		Interrupt disabled/enabled
D_REQ	1		Only on the last TCD of the chain.
START	0		No software request

26.11.7 Use cases and limitations

- In LIN slave mode, the DMA capability can be used only if the ID filtering mode is activated. The number of ID filters enabled must be equal to the number of DMA channels enabled. The correspondence between channel # and ID filter is based on IFMI (identifier filter match index).
- In LIN master mode both the DMA channels (TX and RX) must be enabled in case the DMA capability is required.
- In UART mode the DMA capability can be used only if the UART Tx/Rx buffers are configured as FIFOs.
- DMA and CPU operating modes are mutually exclusive for the data/frame transfer on a UART or LIN node. Once a DMA transfer is finished the CPU can handle subsequent accesses.
- Error management must be always executed via CPU enabling the related error interrupt sources. The DMA capability does not provide support for the error

management. Error management means checking status bits, handling IRQs and potentially canceling DMA transfers.

- The DMA programming model must be coherent with the TCD setting defined in this document.

26.12 Functional description

26.12.1 8-bit timeout counter

26.12.1.1 LIN timeout mode

Resetting the LTOM bit in the LINTCSR enables the LIN timeout mode. The LINOCR becomes read-only, and OC1 and OC2 output compare values in the LINOCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the MME bit in LINCR1), 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).

26.12.1.1.1 LIN Master mode

Field RTO in the LINTOCR 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} (refer to *Figure 326*).

When LINFlexD moves from Break delimiter state to Synch Field state (refer to *Section 26.10.3: 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.

26.12.1.1.2 LIN Slave mode

Field RTO in the LINTOCR 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} (refer to *Figure 326*).

When LINFlexD moves from Break state to Break Delimiter state (refer to *Section 26.10.3: 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.

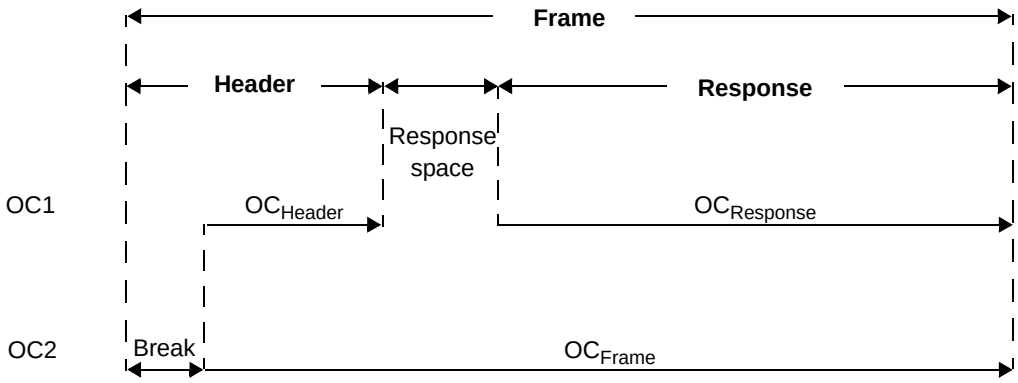


Figure 326. Header and response timeout

26.12.1.2 Output compare mode

Setting the LTOM bit in the LINTCSR 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.

26.12.2 Interrupts

Table 297. LINFlexD interrupt control

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Header Received interrupt	HRF	HRIE	RXI (1)
Data Transmitted interrupt	DTF	DTIE	TXI
Data Received interrupt	DRF	DRIE	RXI
Data Buffer Empty interrupt	DBEF	DBEIE	TXI
Data Buffer Full interrupt	DBFF	DBFIE	RXI

Table 297. LINFlexD interrupt control (continued)

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Wake-up interrupt	WUPF	WUPIE	RXI
LIN State interrupt (2)	LSF	LSIE	RXI
Buffer Overrun interrupt	BOF	BOIE	ERR
Framing Error interrupt	FEF	FEIE	ERR
Header Error interrupt	HEF	HEIE	ERR
Checksum Error interrupt	CEF	CEIE	ERR
Bit Error interrupt	BEF	BEIE	ERR
Output Compare interrupt	OCF	OCIE	ERR
Stuck at Zero interrupt	SZF	SZIE	ERR

1. In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.
2. For debug and validation purposes.

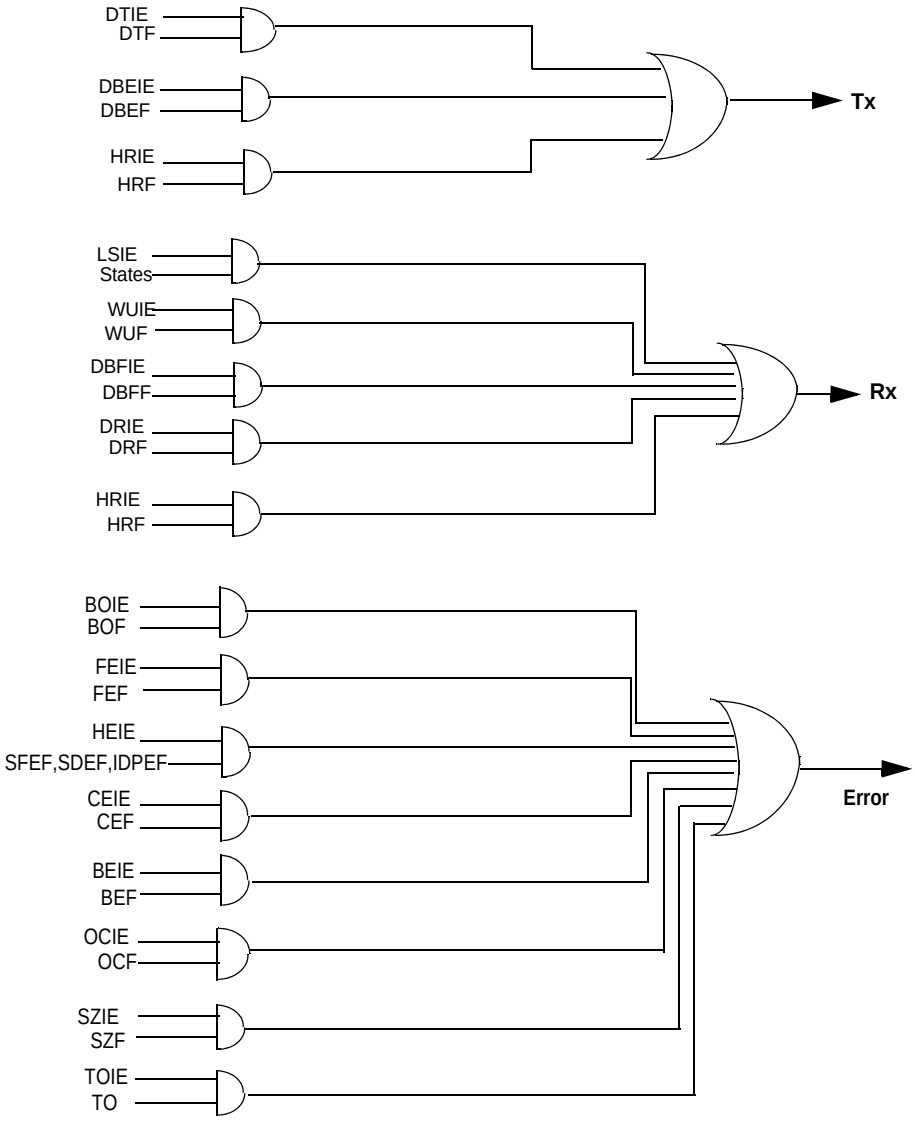


Figure 327. Interrupt diagram

26.12.3 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.

$$\text{Tx/Rx baud} = \frac{f_{ipg_clock_lin}}{(16 * \text{LFDIV})}$$

LFDIV is an unsigned fixed point number. The 20-bit mantissa is coded in the LINIBRR register and the fraction is coded in the LINFBRR register.

The following examples show how to derive LFDIV from LINIBRR and LINFBRR register values:

Example 1

If LINIBRR = 27d and LINFBRR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

Example 2

To program LFDIV = 25.62d,

LINFBRR = 16 * 0.62 = 9.92, nearest real number 10d = Ah

LINIBRR = mantissa(25.620d) = 25d = 19h

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 LINIBRR.

Note: LFDIV must be greater than or equal to 1.5d, for example, LINIBRR = 1 and LINFBRR = 8. Therefore, the maximum possible baud rate is fperiph_set_1_clk / 24.

Table 298. Error calculation for programmed baud rates

Baud rate	$f_{periph_set1_clk}$ = 64 MHz				$f_{periph_set1_clk}$ = 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
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

26.13 Programming considerations

This section describes the various configurations in which the LINFlexD can be used.

26.13.1 Master node

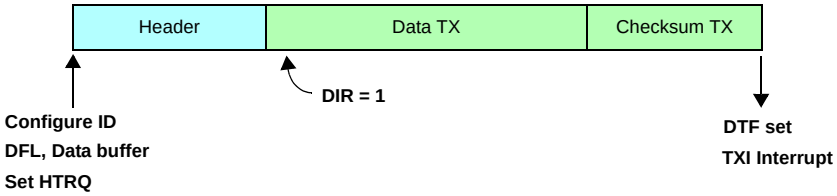


Figure 328. Programming consideration: master node, transmitter

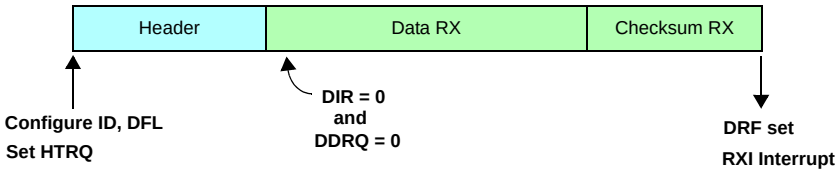


Figure 329. Programming consideration: master node, receiver

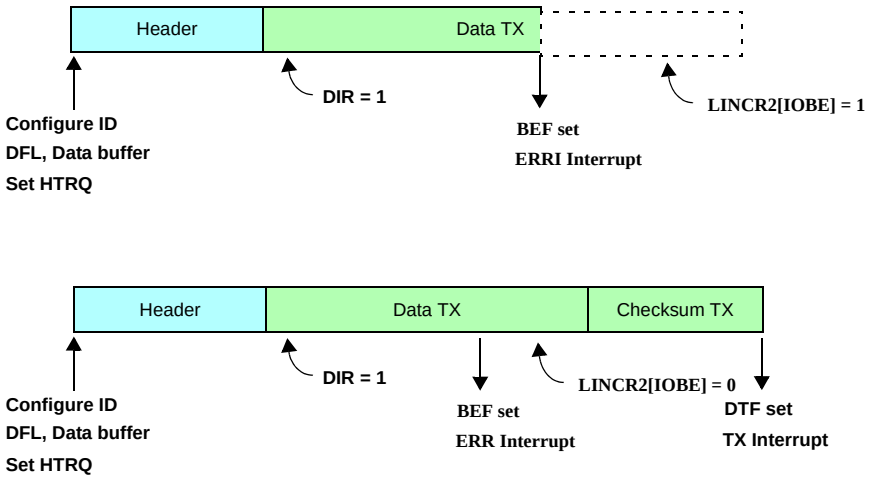


Figure 330. Programming consideration: master node, transmitter, bit error

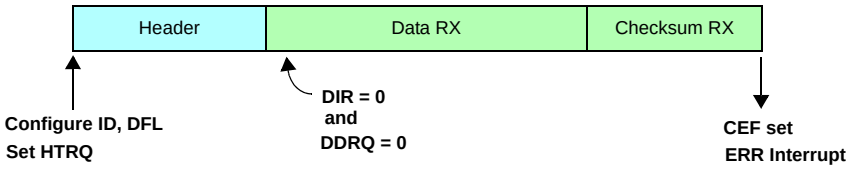


Figure 331. Programming consideration: master node, receiver, checksum error

26.13.2 Slave node

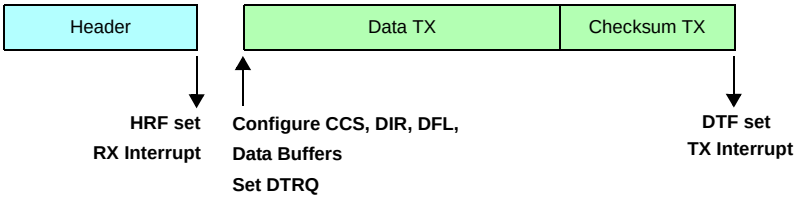


Figure 332. Programming consideration: slave node, transmitter, no filters

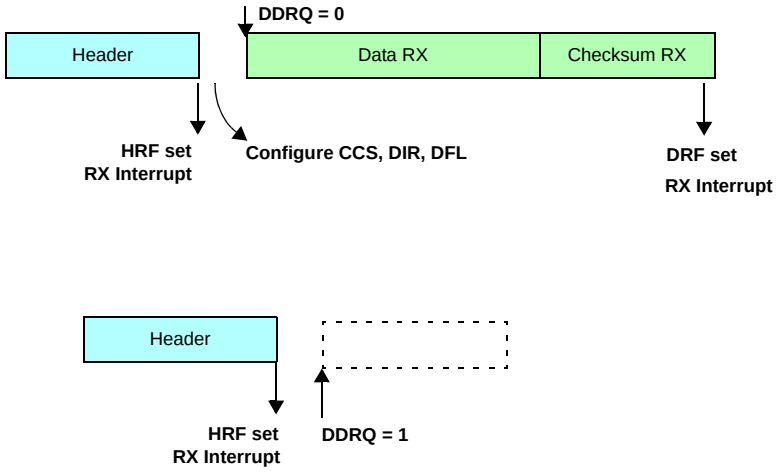


Figure 333. Programming consideration: slave node, receiver, no filters

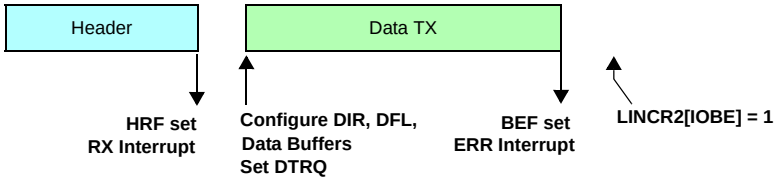


Figure 334. Programming consideration: slave node, transmitter, no filters, bit error

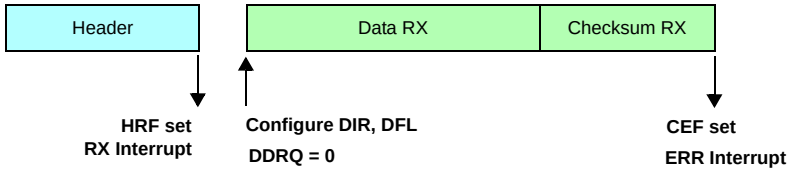
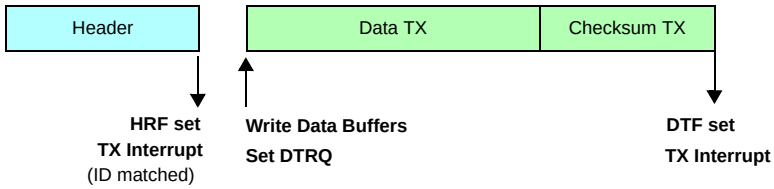


Figure 335. Programming consideration: slave node, receiver, no filters, checksum error



Note: This configuration can be used in case the slave never receives data (for example, as with a sensor).

Figure 336. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter

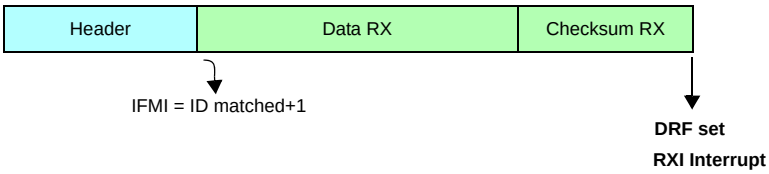


Figure 337. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter

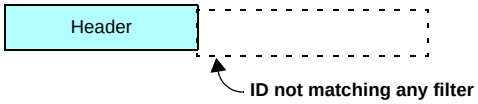
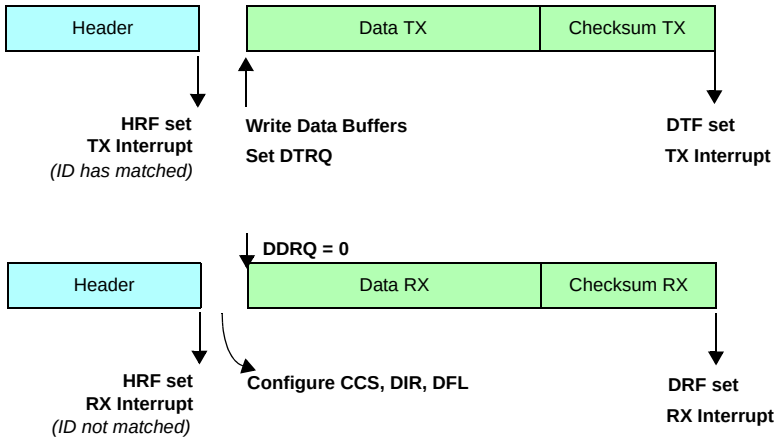


Figure 338. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset



Note: This configuration is used when:
a) All TX IDs are managed by filters
b) The number of other filters is not enough to manage all reception IDs

Figure 339. Programming consideration: slave node, TX filter, BF is set

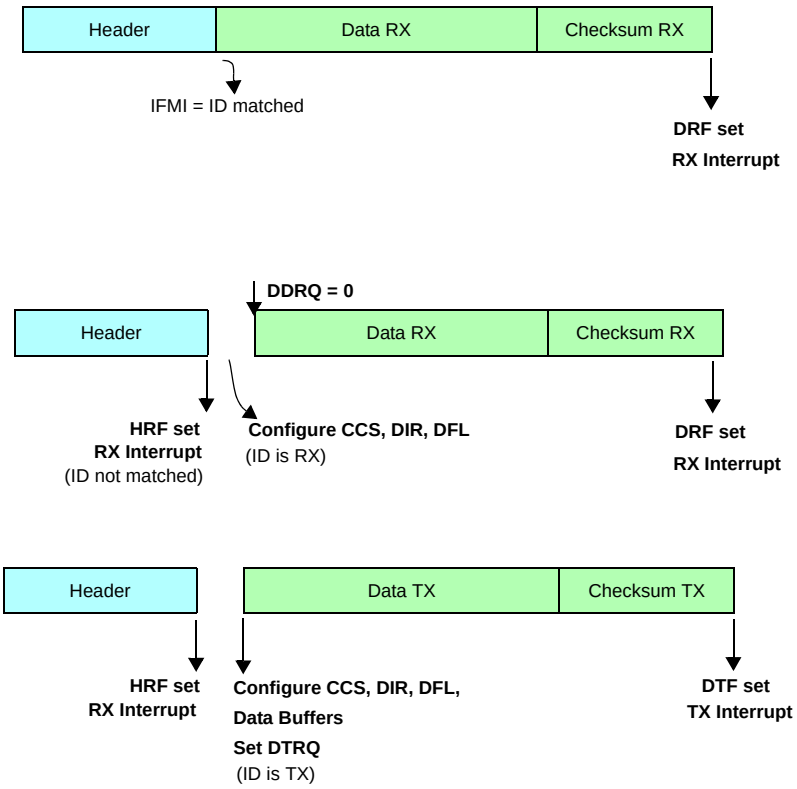
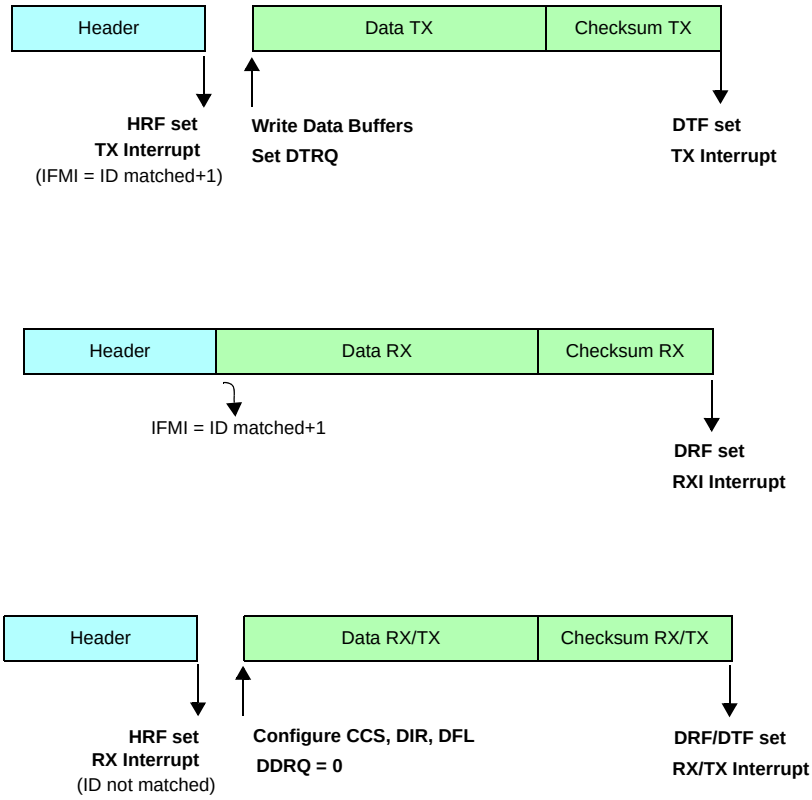


Figure 340. Programming consideration: slave node, RX filter, BF is set



Note: This configuration is used when:
a) The number of filters is not enough
b) Filters are used for most frequently-used IDs to reduce CPU usage

Figure 341. Programming consideration: slave node, TX filter, RX filter, BF is set

26.13.3 Extended frames

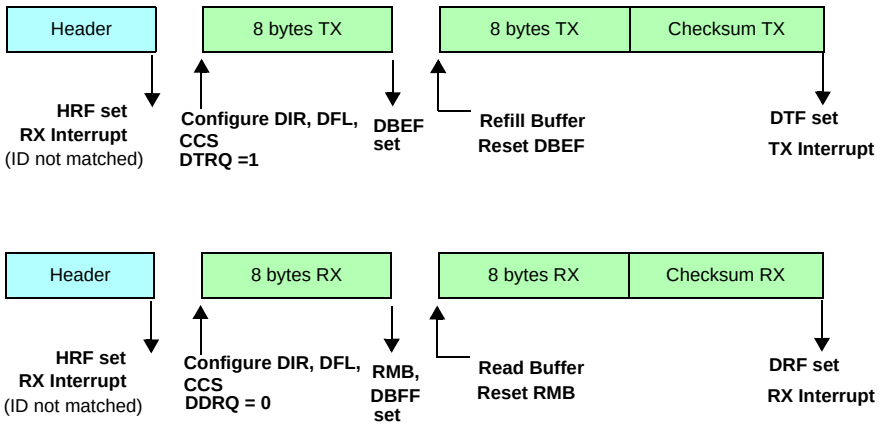


Figure 342. Programming consideration: extended frames

26.13.4 Timeout

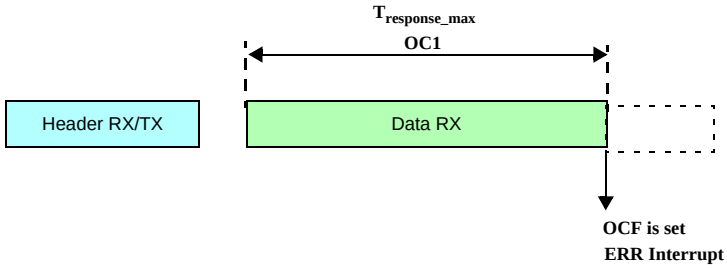


Figure 343. Programming consideration: response timeout(1)

1 Please note that in case of back to back header transmission, LINTCSR[IOT} should be set. If this is not the case, LIN state machine will remain in response state waiting for the response infinitely and the next header will not be recognized. Corresponding error flag in LINESR will be set.

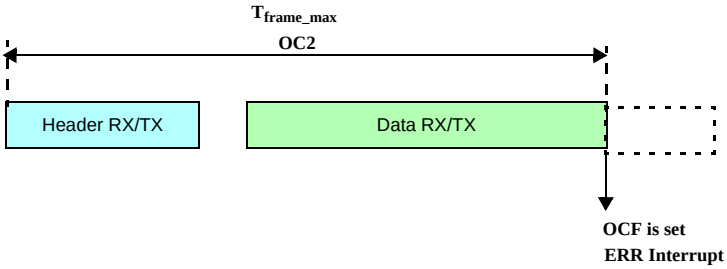


Figure 344. Programming consideration: frame timeout

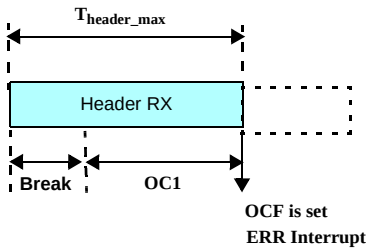


Figure 345. Programming consideration: header timeout

26.13.5 UART mode

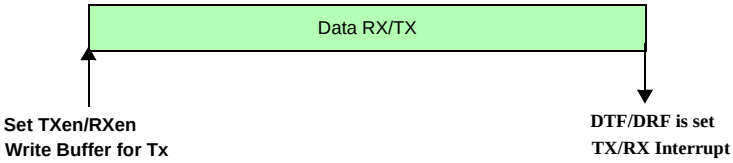


Figure 346. Programming consideration: UART mode

27 FlexCAN

27.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

27.1.1 Device-specific features

The device has six Controller Area Network (FlexCAN) blocks.

- Each block supports 64 Message Buffers (MB).
- DMA support is not provided.
- It is possible to operate the bxcan FlexCAN bit timing logic with either system clock or 4–40 MHz fast external crystal oscillator clock (FXOSC).
- In the case of safe mode entry, the pad associated with CANTX can optionally be put into a high-impedance state (not recessive state)
- Modes of operation:
  - 4 functional modes: Normal (User and Supervisor), Freeze, Listen-Only and Loop-Back
  - 1 low-power mode (Disable mode)
- 1056 bytes (64 MBs) of RAM used for MB storage
- 256 bytes (64 MBs) of RAM used for individual Rx Mask Registers
- Hardware cancellation on Tx message buffers
- *Section 27.4.4.1: Module Configuration Register (MCR)*: Bits 5, 9, 12 and 13 are ‘Reserved’
- *Section 27.4.4.8: Error and Status Register (ESR)*: Bit 31 is ‘Reserved’

27.2 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification [Ref. 1]. A general block diagram is shown in *Figure 347*, 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 sub-modules are described in subsequent sections.

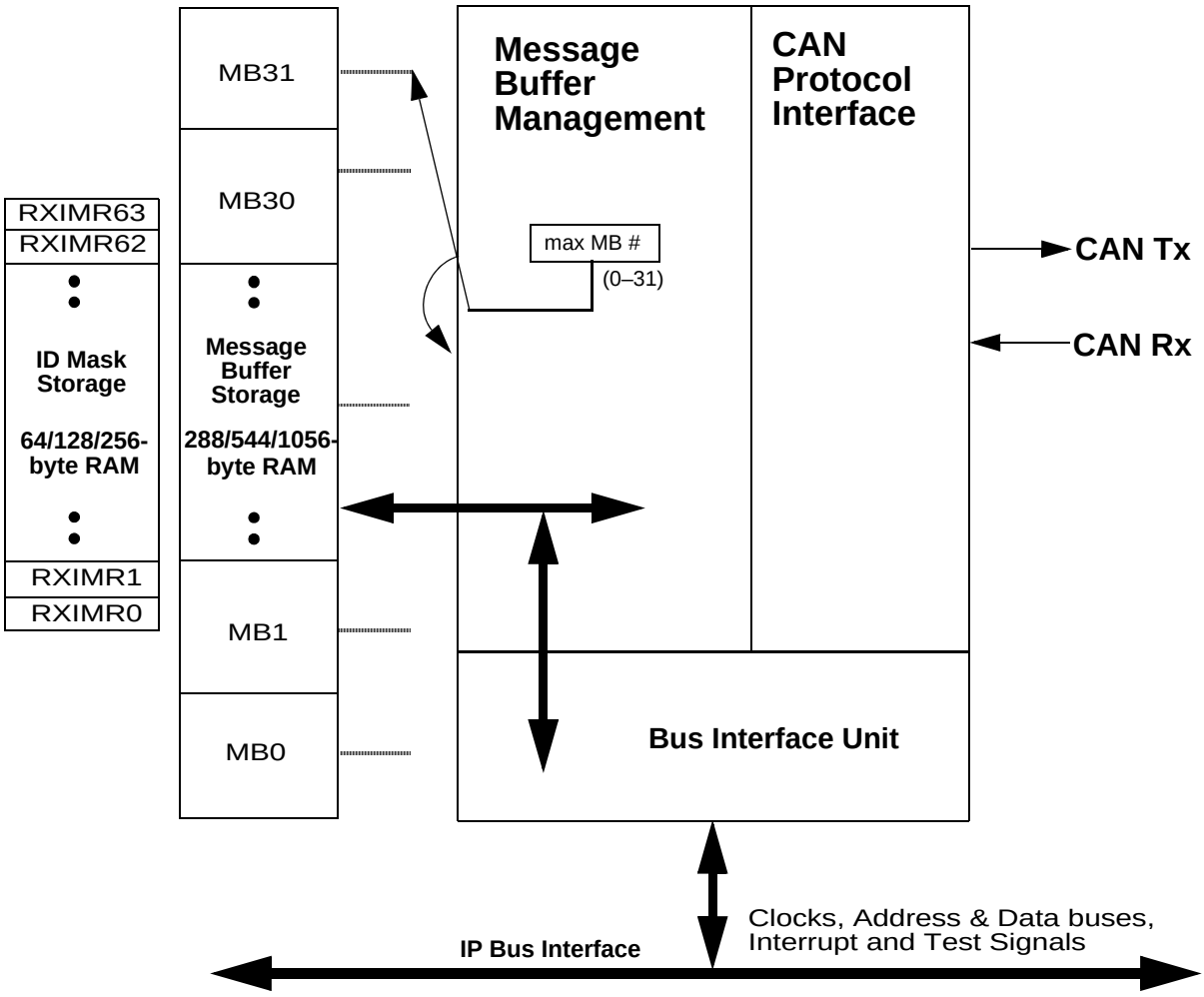


Figure 347. FlexCAN block diagram

27.2.1 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 [Ref. 1], 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 RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) sub-module handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) sub-module 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.

27.2.2 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
  - Zero to eight 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), 544 bytes (32 MBs) or 288 bytes (16 MBs) of RAM used for MB storage
- Includes either 256 bytes (64 MBs), 128 bytes (32 MBs) or 64 bytes (16 MBs) of RAM used for individual Rx Mask Registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard or 32 partial (8 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 RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages

27.2.3 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 Register 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 27.5.9.1: Freeze mode*, for more information.

- Listen-Only mode:

  The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

- Loop-Back mode:

  The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

- Module Disable mode:

  This low power mode is entered when the MCR[MDIS] bit is asserted by the CPU. When disabled, the module requests to disable the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by negating the MDIS bit in the MCR Register. See *Section 27.5.9.2: Module Disable mode*, for more information.

27.3 External signal description

27.3.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 299* and described in more detail in the next subsections.

Table 299. 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.

27.3.2 Signal descriptions

27.3.2.1 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’.

27.3.2.2 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’.

27.4 Memory map/register definition

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 RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

27.4.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 64 MBs capability is shown in *Table 300*.

All registers except for the MCR can be configured to have either supervisor or unrestricted access by programming the MCR[SUPV] bit.

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 RAM (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 300. 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 698*
0x0004	Control Register (CTRL)	*on page 702*
0x0008	Free Running Timer (TIMER)	*on page 705*
0x000C	Reserved	
0x0010	Rx Global Mask (RXGMASK)	*on page 706*
0x0014	Rx Buffer 14 Mask (RX14MASK)	*on page 708*
0x0018	Rx Buffer 15 Mask (RX15MASK)	*on page 708*
0x001C	Error Counter Register (ECR)	*on page 708*
0x0020	Error and Status Register (ESR)	*on page 710*
0x0024	Interrupt Masks 2 (IMASK2)	*on page 712*
0x0028	Interrupt Masks 1 (IMASK1)	*on page 713*
0x002C	Interrupt Flags 2 (IFLAG2)	*on page 714*
0x0030	Interrupt Flags 1 (IFLAG1)	*on page 714*
0x0034–0x007F	Reserved	
0x0080–0x017F	Message Buffers MB0–MB15	—
0x0180–0x027F	Message Buffers MB16–MB31	—
0x0280–0x047F	Message Buffers MB32–MB63	—
0x0480-087F	Reserved	
0x0880-0x08BC	Rx Individual Mask Registers RXIMR0-RXIMR15	*on page 715*
0x08C0-0x08FC	Rx Individual Mask Registers RXIMR16-RXIMR31	*on page 715*
0x0900-0x097C	Rx Individual Mask Registers RXIMR32-RXIMR63	*on page 715*

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 301*. *Table 301* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 301. 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)

27.4.2 Message Buffer Structure

The Message Buffer structure used by the FlexCAN module is represented in *Figure 348*. 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.

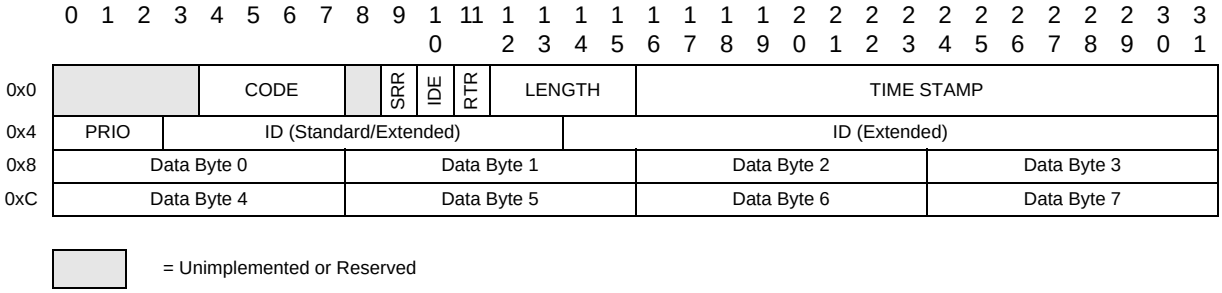


Figure 348. Message Buffer Structure

Table 302. 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 303* and *Table 304*. See *Section 27.5: 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 302. 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 **Note:** Do not configure the last Message Buffer to be RTR frame
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 *Figure 348*). 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 27.5.4: 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 303. 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.
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.
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 304*), the BUSY bit should be ignored upon read,

Table 304. 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.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

27.4.3 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. *Figure 349* 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. *Figure 350* 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 27.5.7: Rx FIFO*, for more information.

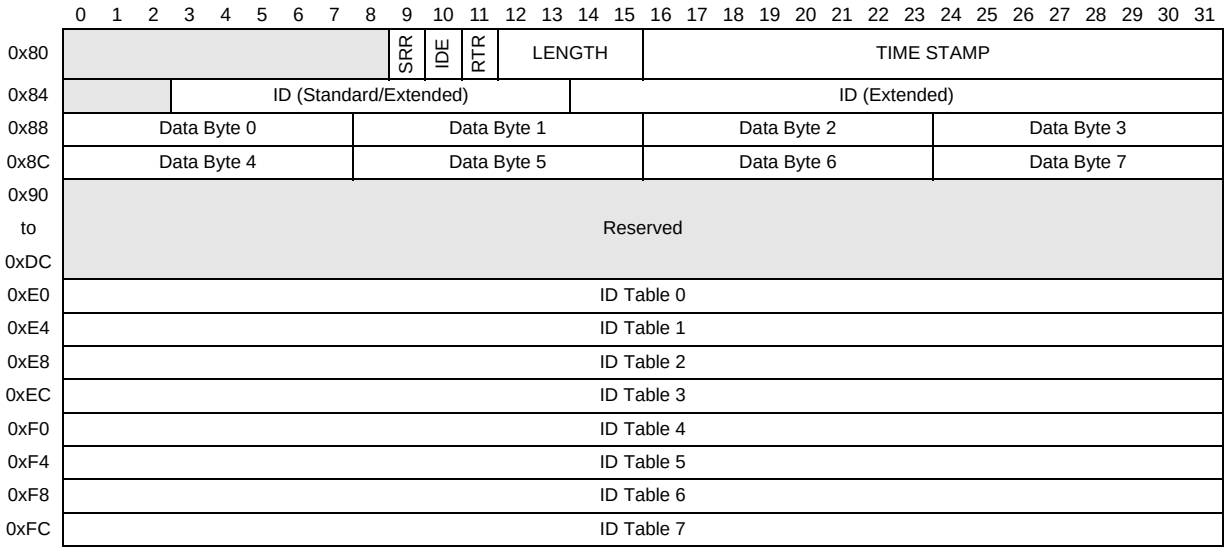


Figure 349. Rx FIFO structure

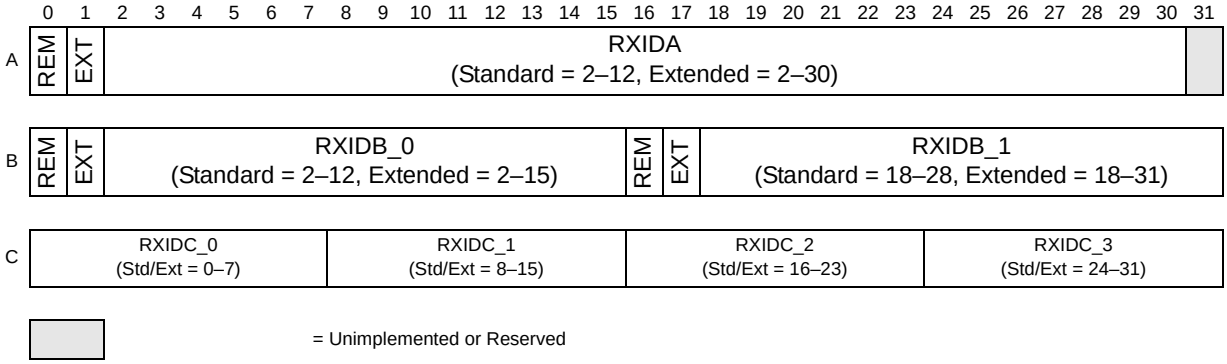


Figure 350. ID Table 0–7

Table 305. 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.

27.4.4 Register descriptions

The FlexCAN registers are described in this section in ascending address order.

27.4.4.1 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.

Offset: 0x0000 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	SUPV	0	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W																
RESET:	1	1	0	1	1	0	0	Note(1)	1	0	0	Note(2)	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	0(3)	0	0	IDAM		0	0	MAXMB					
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 351. Module Configuration Register (MCR)

1. Different on various platforms, but it is always the opposite of the MDIS reset value.
2. Different on various platforms, but it is always the same as the MDIS reset value.
3. This bit must always be written 0

Table 306. 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 sub-modules. This is the only bit in MCR not affected by soft reset. See *Section 27.5.9.2: 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 Register 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 27.4.3: Rx FIFO structure*, and *Section 27.5.7: Rx FIFO*, for more information. This bit must be written in Freeze mode only. 1 = FIFO enabled 0 = FIFO not enabled

Table 306. 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 can not be entered while FlexCAN is in any of the low power modes. See *Section 27.5.9.1: 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 in Disable Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 1 = FlexCAN module is in Disable Mode or Freeze Mode 0 = FlexCAN module is 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 Register, 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 any of the low power modes. 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 300* 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 any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See *Section 27.5.9.1: Freeze mode*, for more information. 1 = FlexCAN in Freeze Mode, prescaler stopped 0 = FlexCAN not in Freeze Mode, prescaler running
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 300*. 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

Table 306. MCR field descriptions (continued)

Field	Description
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 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 27.5.9.2: Module Disable mode*, for more information. 1 = FlexCAN is in Disable Mode 0 = FlexCAN not in any low-power 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. FlexCAN on this device supports individual Rx ID masking using RXIMR0─RXIMR63; setting this bit enables individual Rx ID masking. When this bit is cleared FlexCAN uses a backwards compatible masking scheme with RXGMASK, RX14MASK and RX15MASK and 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 cleared allowing legacy software to work without modification. This bit must be written in Freeze mode only. 0 = Individual Rx masking and queue feature are disabled. 1 = Individual Rx masking and queue feature are enabled.
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

Table 306. MCR field descriptions (continued)

Field	Description
IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 307*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 27.4.3: 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 307. IDAM coding

IDAM	Format	Explanation
0b00	A	One full ID (standard or extended) per filter element.
0b01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
0b10	C	Four partial 8-bit IDs (standard or extended) per filter element.
0b11	D	All frames rejected.

Note: After resetting PWDN bit, wait for the power up time suggested by ADC power up parameter in the DS (tADC_PU) before any conversion is initiated

27.4.4.2 Control Register (CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK and BOFF_REC bits, that can be accessed at any time.

Offset: 0x0004 Access: Read/write

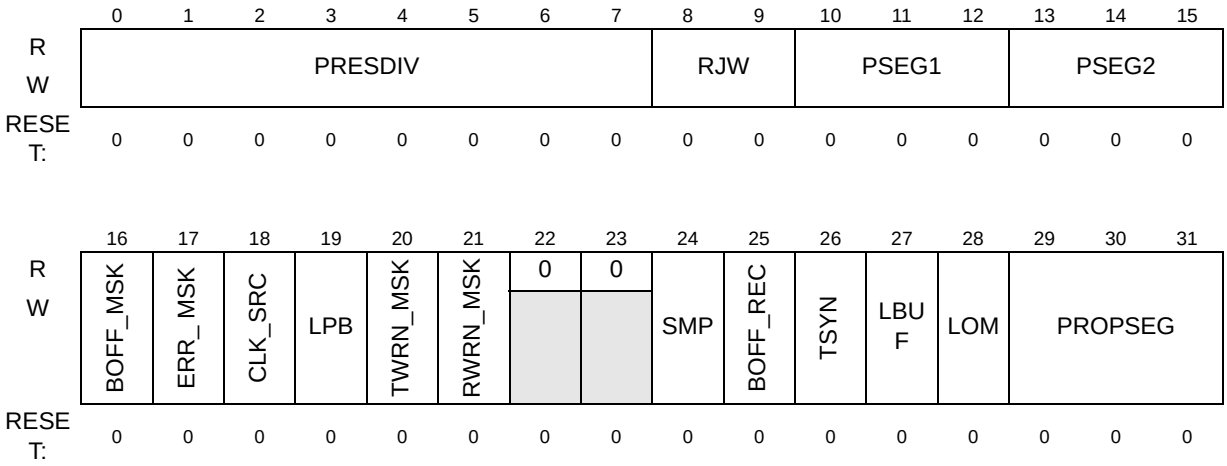


Figure 352. Control Register (CTRL)

Table 308. CTRL field descriptions

Field	Description
PRESDIV	Prescaler Division Factor This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to *Section 27.5.8.4: Protocol timing*. This bit must be written in Freeze mode only. Sclock frequency = CPI clock frequency / (PRESDIV + 1)
RJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta(1) that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. This bit must be written in Freeze mode only. Resync Jump Width = RJW + 1.
PSEG1	Phase Segment 1 This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only. Phase Buffer Segment 1 = (PSEG1 + 1) x Time-Quanta.
PSEG2	Phase Segment 2 This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. 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
ERR_MSK	Error Mask This bit provides a mask for the Error Interrupt. 1 = Error interrupt enabled 0 = Error interrupt disabled

Table 308. CTRL field descriptions (continued)

Field	Description
CLK_SRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit must only be changed while the module is in Disable Mode. See *Section 27.5.8.4: 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:** In order to guarantee reliable operation, the selected CAN Protocol Interface (CPI) clock should not be faster than the peripheral clock.
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
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
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 2 preceding samples, a majority rule is used 0 = Just one sample is used to determine the bit value

Table 308. 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 re-synchronize 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 (i.e., 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 [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 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 3-bit 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.

27.4.4.3 Free Running Timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU.

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.

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 W	TIMER															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 353. Free Running Timer (TIMER)

Table 309. TIMER field descriptions

Field	Description
TIMER	Free-running timer counter. The timer starts from 0x0000 after reset, counts linearly to 0xFFFF, and wraps around.

27.4.4.4 Rx Global Mask (RXGMASK)

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RXGMASK Register to have no effect on the module operation.

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.

See *Section 27.5.7: 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.

During CAN messages reception by FlexCAN, the RXGMASK (Rx Global Mask) is used as acceptance mask for most of the Rx Message Buffers (MB). When the FIFO Enable bit in the FlexCAN Module Configuration Register (CANx_MCR[FEN], bit 2) is set, the RXGMASK also applies to most of the elements of the ID filter table. However, there is a misalignment between the position of the ID field in the Rx MB and in RXIDA, RXIDB and RXIDC fields of

the ID Tables. In fact RXIDA filter in the ID Tables is shifted one bit to the left from Rx MBs ID position as shown below:

- Rx MB ID = bits 3–31 of ID word corresponding to message ID bits 0–28
- RXIDA = bits 2–30 of ID Table corresponding to message ID bits 0–28

The mask bits one-to-one correspondence occurs with the filters bits, not with the incoming message ID bits. This leads the RXGMASK to affect Rx MB and Rx FIFO filtering in different ways.

For example, if the user intends to mask out the bit 24 of the ID filter of Message Buffers then the RXGMASK will be configured as 0xffff_ffef. As result, bit 24 of the ID field of the incoming message will be ignored during filtering process for Message Buffers. This very same configuration of RXGMASK would lead bit 24 of RXIDA to be "don`t care" and thus bit 25 of the ID field of the incoming message would be ignored during filtering process for Rx FIFO.

Similarly, both RXIDB and RXIDC filters have multiple misalignments with regards to position of ID field in Rx MBs, which can lead to erroneous masking during filtering process for either Rx FIFO or MBs.

RX14MASK (Rx 14 Mask) and RX15MASK (Rx 15 Mask) have the same structure as the RXGMASK. This includes the misalignment problem between the position of the ID field in the Rx MBs and in RXIDA, RXIDB and RXIDC fields of the ID Tables.

Therefore it is recommended that one of the following actions be taken to avoid problems:

- Do not enable the RxFIFO. If CANx_MCR[FEN]=0 then the Rx FIFO is disabled and thus the masks RXGMASK, RX14MASK and RX15MASK do not affect it.
- Enable Rx Individual Mask Registers. If the Backwards Compatibility Configuration bit in the FlexCAN Module Configuration Register (CANx_MCR[BCC], bit 15) is set then the Rx Individual Mask Registers (RXIMR0–63) are enabled and thus the masks RXGMASK, RX14MASK and RX15MASK are not used.
- Do not use masks RXGMASK, RX14MASK and RX15MASK (i.e., let them in reset value which is 0xffff_ffff) when CANx_MCR[FEN]=1 and CANx_MCR[BCC]=0. In this case, filtering processes for both Rx MBs and Rx FIFO are not affected by those masks.
- Do not configure any MB as Rx (i.e., let all MBs as either Tx or inactive) when CANx_MCR[FEN]=1 and CANx_MCR[BCC]=0. In this case, the masks RXGMASK, RX14MASK and RX15MASK can be used to affect ID Tables without affecting filtering process for Rx MBs.

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

Figure 354. Rx Global Mask Register (RXGMASK)

Table 310. RXGMASK field descriptions

Field	Description
MI*n*	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”

27.4.4.5 Rx 14 Mask (RX14MASK)

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. 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.

See *Section 27.5.7: Rx FIFO*, for important details on usage of RX14MASK 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.

- Address Offset: 0x14
- Reset Value: 0xFFFF_FFFF

27.4.4.6 Rx 15 Mask (RX15MASK)

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. 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 RXG15MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

Refer to *Section 27.5.7: Rx FIFO*, for important details on usage of RX15MASK 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.

- Address Offset: 0x18
- Reset Value: 0xFFFF_FFFF

27.4.4.7 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.

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

Figure 355. Error Counter Register (ECR)

Table 311. ECR field descriptions

Field	Description
RX_ERROR_ COUNTER	Receive Error Counter. See the text of this section for a detailed description of this field and how it interacts with TX_ERROR_COUNTER.
TX_ERROR_ COUNTER	Transmit Error Counter. See the text of this section for a detailed description of this field and how it interacts with RX_ERROR_COUNTER.

27.4.4.8 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, and ERR_INT, that are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See *Section 27.5.10: Interrupts*, for more details.

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

Figure 356. Error and Status Register (ESR)

Table 312. ESR field descriptions

Field	Description
TWRN_INT	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

Table 312. ESR field descriptions (continued)

Field	Description
RWRN_INT	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_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.
BIT0_ERR	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	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	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	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	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 Counter This bit indicates when repetitive errors are occurring during message reception. 1 = Rx_Err_Counter ≥ 96 0 = No such occurrence

Table 312. ESR field descriptions (continued)

Field	Description
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)
FLT_CONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in *Table 313*. 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 313. Fault confinement state

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

27.4.4.9 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).

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

Figure 357. Interrupt Masks 2 Register (IMASK2)

Table 314. MASK2 field descriptions

Field	Description
BUF*n*M	Buffer MB_n 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.

27.4.4.10 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).

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

Figure 358. Interrupt Masks 1 Register (IMASK1)

Table 315. IMASK1 field descriptions

Field	Description
BUF*n*M	Buffer MB_n 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.

27.4.4.11 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.

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

Figure 359. Interrupt Flags 2 Register (IFLAG2)

Table 316. IFLAG2 field descriptions

Field	Description
BUF*n*I	Buffer MB_n 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

27.4.4.12 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 MCR[FEN] bit 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.

Offset: 0x002C 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

Figure 360. Interrupt Flags 1 Register (IFLAG1)

Table 317. IFLAG1 field descriptions

Field	Description
BUF31I–BUF8I	Buffer MB_n 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

27.4.4.13 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 eight elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze mode. Out of Freeze mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the MCR Register is negated, any read or write operation to these registers results in access error.

Offsets: 0x0880–0x097F (64 registers0 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

	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

Figure 361. Rx Individual Mask Registers (RXIMR0–RXIMR63)

Table 318. RXIMR0–RXIMR63 field descriptions

Field	Description
MI*n*	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”

27.5 Functional description

27.5.1 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 27.4.2: 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 303*). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to *Table 304*). 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 27.5.6.1: Message Buffer Deactivation*).

27.5.2 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. *Figure 348* 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.

27.5.3 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

1. If the MB is active (transmission pending), write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see Section 1.5.6.2, “Message Buffer Deactivation)
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step four (see *Table 303* and *Table 304* in *Section 27.4.2: Message Buffer Structure*).

27.5.4 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[j] or the lowest MB number or the highest

j. 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.

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”. 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.

27.5.5 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:

1. If the MB has a pending transmission, write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see *Section 27.5.6.1: Message Buffer Deactivation*). If the MB already programmed as a receiver, just write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive.
2. Write the ID word
3. Write ‘0100’ to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see *Table 303* and *Table 304* in *Section 27.4.2: Message Buffer Structure*)
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 27.5.6: Data coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in *Table 303*. 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 27.5.7: Rx FIFO*). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

27.5.6 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in *Section 27.5.3: Transmit process* and *Section 27.5.5: 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.

27.5.6.1 Message Buffer Deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated.

27.5.6.2 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(k) (‘0100’). Also, Tx MBs can not be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages

into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

Note: If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

27.5.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see *Section 27.4.3: 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 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 five 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 eight 32-bit registers that can be configured to one of the following formats (see also *Section 27.4.3: 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

k. 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.

Note: A chosen format is applied to all eight 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.

27.5.8 CAN Protocol Related Features

27.5.8.1 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).

27.5.8.2 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

27.5.8.3 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 27.4.4.2: Control Register (CTRL)*.

27.5.8.4 Protocol timing

Figure 362 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK_SRC) in the CTRL Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

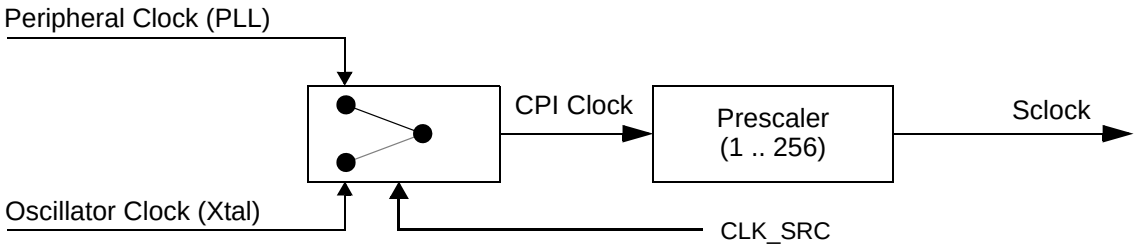


Figure 362. CAN Engine Clocking Scheme

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 PLL generated clocks.

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 27.4.4.2: 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[l] (reference *Figure 363* and *Table 319*):

- 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

l. 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.

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

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

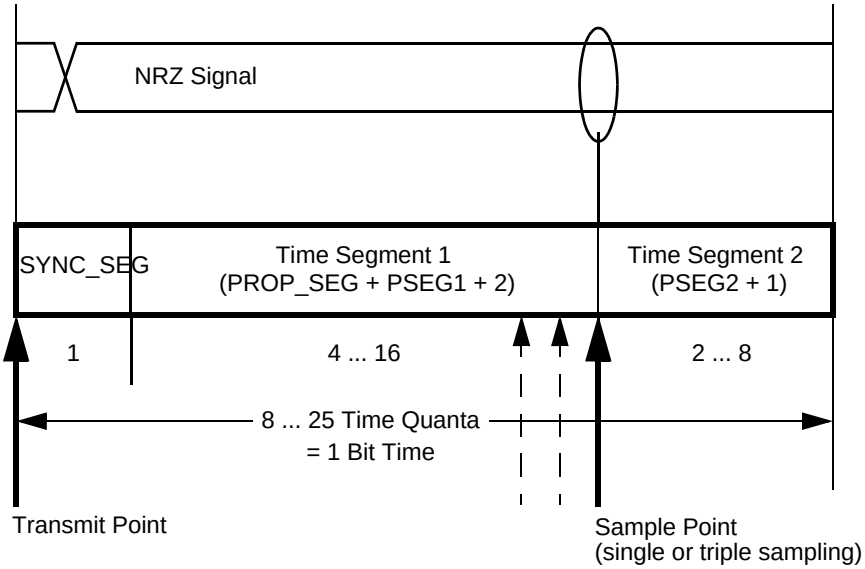


Figure 363. Segments within the Bit Time

Table 319. 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 320 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 320. Bosch CAN 2.0B standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Re-synchronization 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

Table 320. Bosch CAN 2.0B standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
8–15	7	1–4
9–16	8	1–4

Note: Other combinations of Time Segment 1 and Time Segment 2 can be valid. 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.

27.5.8.5 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 364*.

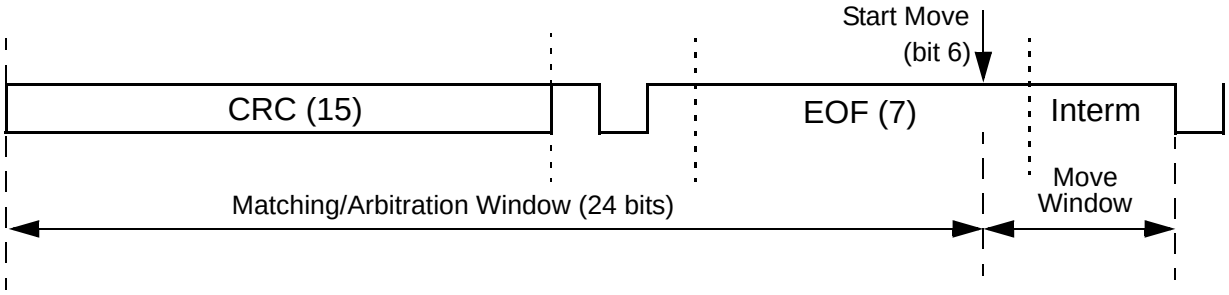


Figure 364. Arbitration, Match and Move Time Windows

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 320*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, i.e., the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in *Table 321*

Table 321. 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 321* 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.

27.5.9 Modes of operation details

27.5.9.1 Freeze mode

This mode is entered by asserting the HALT bit in the MCR Register 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 Register and the module is not in a 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 Register
- 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.

27.5.9.2 Module Disable mode

This low power mode is entered when the MCR[MDIS] bit is asserted. If the module is disabled during Freeze Mode, it requests to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM_ACK bit.

27.5.10 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, 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 27.4.4.12: 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, 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.

27.5.11 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 RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 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 RAM while FlexCAN is transmitting and receiving CAN frames.

27.6 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

27.6.1 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 300* 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 can not be applied while clocks are shut down in any of the low power modes. 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 un-synchronized to the CAN bus, the HALT and FRZ bits in MCR Register are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the MCR Register 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 27.5.9.1: 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 local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers

- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in CTRL Register (for Bus Off and Error interrupts)
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

27.6.2 FlexCAN Addressing and RAM size configurations

There are three RAM 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 Register. 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.

28 Deserial Serial Peripheral Interface (DSPI)

28.1 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.

SPC564Bxx and SPC56ECxx implements eight DSPI modules. 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, and so on.

Figure 365 shows a block diagram of the Deserial Serial Peripheral Interface (DSPI) block.

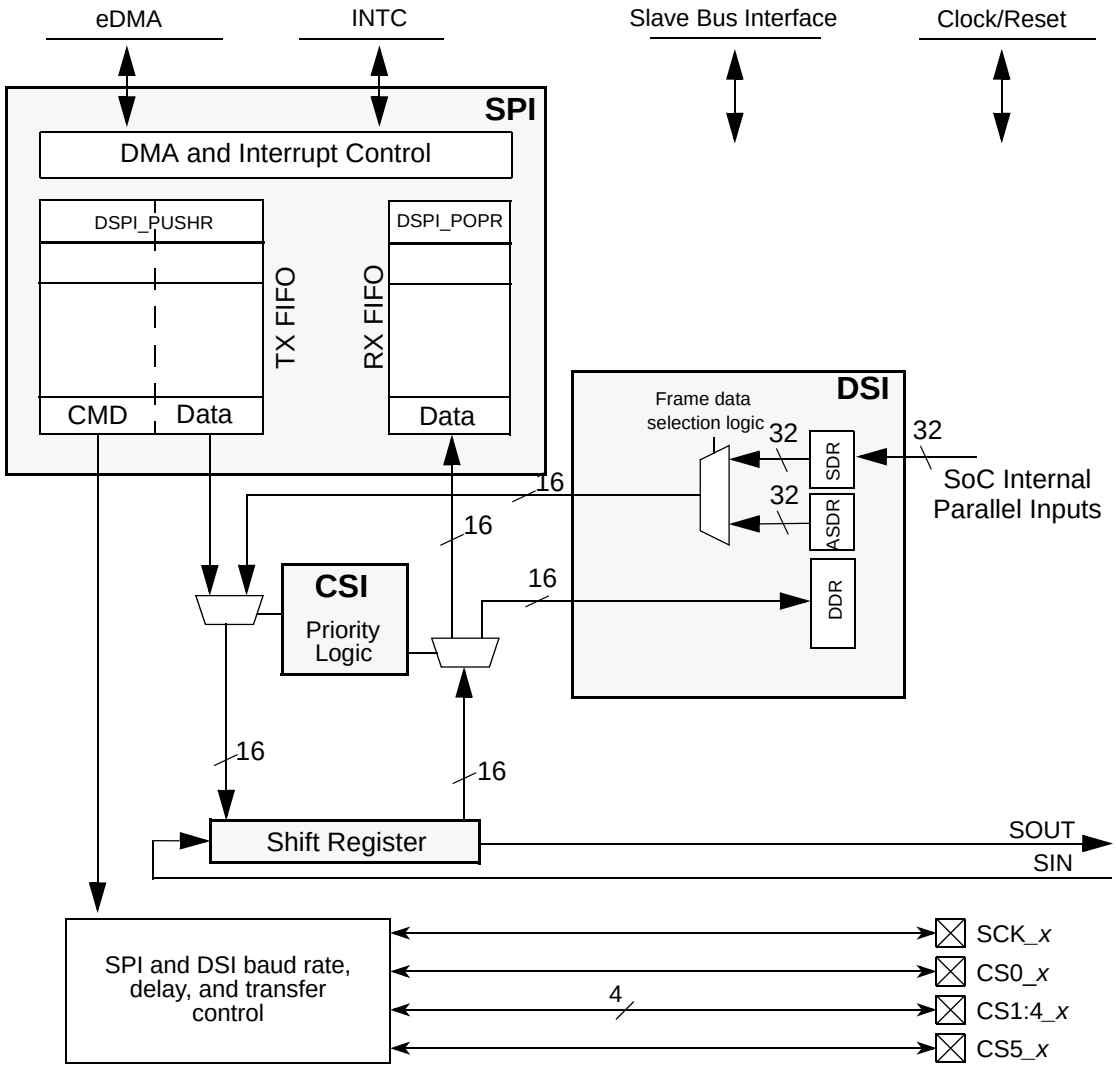


Figure 365. DSPI block diagram

28.1.1 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave modes
  - Data streaming operation in the slave mode with continuous slave selection
- Buffered transmit operation using the TX FIFO with 4 entries
- Buffered receive operation using the RX FIFO with 4 entries
- TX and RX FIFOs can be disabled individually for low-latency updates to SPI queues
- Visibility into TX and RX FIFOs for ease of debugging
- Programmable transfer attributes on a per-frame basis:
  - Parameterized number of transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Various programmable delays
  - Programmable serial frame size of 4 to 32 bits, expandable by software control
  - Continuously held chip select capability
  - Parity control
- Upto six Peripheral Chip Selects, expandable with external demultiplexer
- Deglitching support for Peripheral Chip Select with external demultiplexer
- DMA support for adding entries to TX FIFO and removing entries from RX FIFO:
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 7 Interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - Attempt to transmit with an empty Transmit FIFO (TFUF)
  - RX FIFO is not empty (RFDF)
  - Frame received while Receive FIFO is full (RFOF)
  - SPI Mode Parity Error (SPEF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Power-saving architectural features
  - Support for stop mode

The DSPI also supports pin reduction through serialization and deserialization if enabled for the module.

- 2 Interrupt conditions:
  - Deserialized data matches pre-programmed pattern (DDIF)
  - DSI Mode Parity Error (DPEF)
- Two sources of serialized data:
  - DSPI memory-mapped register
  - Parallel Input signals

  - Programmable selection of source data on bit basis
- Transfer initiation conditions:
  - Continuous
  - Edge sensitive hardware trigger
  - Change in data
- Support for serial chaining of 2 DSPI modules inside the SoC
  - Support DSPI serial chaining between DSPI 0 and DSPI 1
  - Support DSPI serial chaining between DSPI 2 and DSPI 3
- Pin serialization with interleaved SPI frames for control and diagnostics
- Support for the downstream Micro Second Channel with Timed Serial Bus (TSB) configuration

28.1.2 DSPI configurations

The DSPI module can operate in three configurations: SPI, DSI and CSI.

28.1.2.1 SPI configuration

The SPI Configuration allows the DSPI to send and receive serial data. This configuration allows the DSPI to operate as a basic SPI block with internal FIFOs supporting external queues operation. Transmit data and received data reside in separate FIFOs. The host CPU or a DMA controller read the received data from the receive FIFO and write transmit data to the transmit FIFO.

For queued operations the SPI queues can reside in system RAM, external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished by a DMA controller or host CPU. *Figure 366* shows a system example with DMA, DSPI and external queues in system RAM.

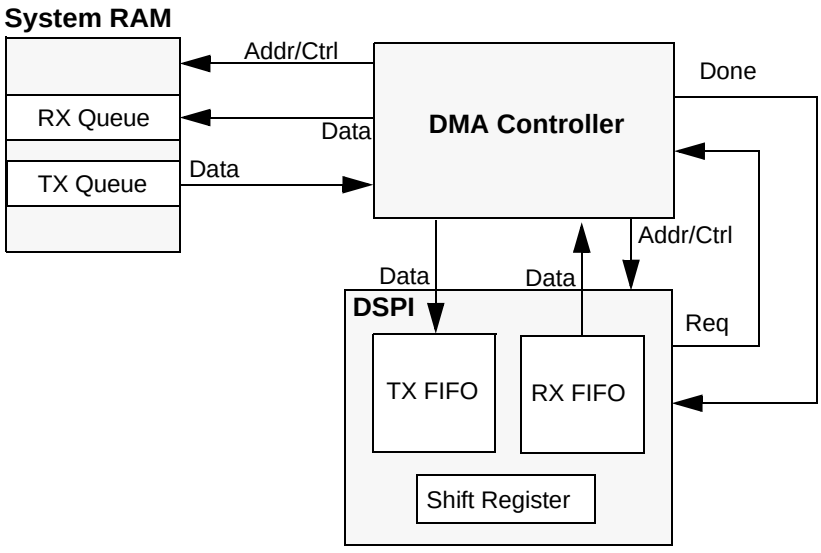


Figure 366. DSPI with Queues and DMA

28.1.2.2 DSI configuration

In the DSI Configuration, the DSPI serializes up to 32 Parallel Input signals or register bits. The data is transferred using a SPI-like protocol.

TSB mode, detailed on *Section 28.4.9: Timed Serial Bus (TSB)*, provides the Micro Second downstream Channel support (MSC), serializing from 4 to 32 Parallel Input signals or register bits.

28.1.2.3 CSI configuration

The CSI configuration is a combination of the SPI and DSI configurations. In this configuration the DSPI interleaves DSI data frames with SPI data frames. Interleaving is done on the frame boundaries. In this configuration, transmission of SPI data has higher priority than DSI data. TSB mode is also operational in CSI configuration.

28.1.3 Modes of operation

The DSPI supports the following modes of operation that can be divided into two categories;

- Module-specific modes:
  - Master mode
  - Slave mode
  - Module disable mode
- SoC-specific modes:
  - External stop mode
  - Debug mode

The DSPI enters module-specific modes when the host writes a DSPI register. The SoC-specific modes are controlled by signals, external to the DSPI. The SoC-specific modes are modes that the entire SoC may enter in parallel to the DSPI block-specific modes.

28.1.3.1 Master Mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK signal and the CS*n* signals are controlled by the DSPI and configured as outputs.

28.1.3.2 Slave Mode

The slave mode allows the DSPI to communicate with SPI/DSI bus masters. In this mode, the DSPI responds to externally controlled serial transfers. The SCK signal and the CS0_*x* signal are configured as inputs and driven by a SPI bus master.

28.1.3.3 Module Disable Mode

The module disable mode can be used for SoC power management. The clock to the non-memory mapped logic in the DSPI can be stopped while in the module disable mode.

28.1.3.4 External Stop Mode

The external stop mode is used for SoC power management. The DSPI supports the Peripheral Bus stop mode mechanism. When a request is made to enter external stop mode, the DSPI block acknowledges the request and completes the transfer in progress. When the DSPI reaches the frame boundary, it signals that the system clock to the DSPI module may be shut off.

28.1.3.5 Debug Mode

The debug mode is used for system development and debugging. The DSPI_MCR[FRZ] bit controls DSPI behavior in the debug mode. If the bit is set, the DSPI stops all serial transfers, when the SoC in the debug mode. If the bit is cleared the SoC debug mode has no effect on the DSPI.

28.2 External signal description

28.2.1 Overview

Table 322 lists off-chip DSPI signals.

Table 322. 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
CS4_x	Output	Peripheral Chip Select 4	Master Trigger
CS5_x	Output	Peripheral Chip Select 5 / Peripheral Chip Select Strobe	Unused
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)

Table 323. DSPI configuration

Feature	DSPI 0	DSPI 1	DSPI 2	DSPI 3	DSPI 4	DSPI 5	DSPI 6	DSPI 7
No of CS supported	6	5	4	2	2	3	4	4
TX Fifo depth	4	4	4	4	4	4	4	4
RX Fifo depth	4	4	4	4	4	4	4	4
CTAR value	6	6	6	6	6	6	6	6

28.2.2 Detailed signal description

28.2.2.1 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.

28.2.2.2 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.

28.2.2.3 Peripheral Chip Select 4 (CS4_x)

CS4_x is a peripheral chip select output signal in master mode.

28.2.2.4 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.

28.2.2.5 Serial Input (SIN)

SIN is a serial data input signal.

28.2.2.6 Serial Output (SOUT)

SOUT is a serial data output signal.

28.2.2.7 Serial Clock (SCK)

SCK is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK is an input from an external bus master.

28.3 Memory map and register definition

28.3.1 Memory map

Register accesses to memory addresses that are reserved or undefined result in a transfer error. Write access to the DSPI_POPR register also result in a transfer error.

Table 324 shows the DSPI memory map.

Table 324. DSPI memory map

Base addresses: 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2) 0xFFF9_C000 (DSPI_3) 0xFFFA_0000 (DSPI_4) 0xFFFA_4000 (DSPI_5) 0xFFFA_8000 (DSPI_6) 0xFFFA_C000 (DSPI_7)		
Address offset	**Register**	**Location**
0x0000	DSPI module configuration register (DSPI_MCR)	*on page 737*
0x0004	Reserved	
0x0008	DSPI transfer count register (DSPI_TCR)	*on page 739*
0x000C	DSPI clock and transfer attributes register 0 (DSPI_CTAR0)	*on page 740*
0x0010	DSPI clock and transfer attributes register 1 (DSPI_CTAR1)	*on page 740*
0x0014	DSPI clock and transfer attributes register 2 (DSPI_CTAR2)	*on page 740*
0x0018	DSPI clock and transfer attributes register 3 (DSPI_CTAR3)	*on page 740*
0x001C	DSPI clock and transfer attributes register 4 (DSPI_CTAR4)	*on page 740*
0x0020	DSPI clock and transfer attributes register 5 (DSPI_CTAR5)	*on page 740*
0x0024 – 0x0028	Reserved	
0x002C	DSPI status register (DSPI_SR)	*on page 745*
0x0030	DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)	*on page 747*
0x0034	DSPI push TX FIFO register (DSPI_PUSHR)	*on page 749*
0x0038	DSPI pop RX FIFO register (DSPI_POPR)	*on page 751*
0x003C	DSPI transmit FIFO register 0 (DSPI_TXFR0)	*on page 751*
0x0040	DSPI transmit FIFO register 1 (DSPI_TXFR1)	*on page 751*
0x0044	DSPI transmit FIFO register 2 (DSPI_TXFR2)	*on page 751*
0x0048	DSPI transmit FIFO register 3 (DSPI_TXFR3)	*on page 751*
0x004C	DSPI transmit FIFO register 4 (DSPI_TXFR4)	*on page 751*
0x0050	DSPI transmit FIFO register 5 (DSPI_TXFR5)	*on page 751*
0x0054 – 0x0078	Reserved	
0x007C	DSPI receive FIFO register 0 (DSPI_RXFR0)	*on page 752*
0x0080	DSPI receive FIFO register 1 (DSPI_RXFR1)	*on page 752*
0x0084	DSPI receive FIFO register 2 (DSPI_RXFR2)	*on page 752*
0x0088	DSPI receive FIFO register 3 (DSPI_RXFR3)	*on page 752*
0x008C – 0x00B8	Reserved	
0x00BC	DSPI DSI Configuration Register (DSPI_DSICR)	*on page 753*
0x00C0	DSPI DSI Serialization Data Register (DSPI_SDR)	*on page 755*

Table 324. DSPI memory map

Base addresses: 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2) 0xFFF9_C000 (DSPI_3) 0xFFFA_0000 (DSPI_4) 0xFFFA_4000 (DSPI_5) 0xFFFA_8000 (DSPI_6) 0xFFFA_C000 (DSPI_7)		
Address offset	**Register**	**Location**
0x00C4	DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)	*on page 755*
0x00C8	DSPI DSI Transmit Comparison Register (DSPI_COMPR)	*on page 756*
0x00CC	DSPI DSI Deserialization Data Register (DSPI_DDR)	*on page 756*
0x00D0	DSPI DSI TSB Configuration Register 1 (DSPI_DSICR1)	*on page 756*
0x00D4	DSPI DSI Serialization Source Select Register (DSPI_SSR)	*on page 756*
0x00D8 – 0x00E4	Reserved	
0x00E8	DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)	*on page 758*
0x00EC	DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)	*on page 759*

28.3.2 Register descriptions

28.3.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits, which configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR are allowed to be changed, while the DSPI is in the Running state.

Offset: 0x0000 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	0	0	SMPL_PT		0	0	0	0	0	0	PES	HALT
W					CLR_TXF	CLR_RXF										
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 367. DSPI Module Configuration Register (DSPI_MCR)

Table 325. DSPI_MCR Field Descriptions

Field	Description
MSTR	Master/Slave Mode Select. The MSTR bit configures the DSPI for either master mode or slave mode. 0 DSPI is in slave mode 1 DSPI is in master mode
CONT_SCKE (1)	Continuous SCK Enable. The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See *Section 28.4.7: Continuous Serial Communications Clock*, for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
DCONF	DSPI Configuration. The DCONF field selects between the three different configurations of the DSPI: 00 SPI 01 DSI 10 CSI 11 Reserved
FRZ	Freeze. The FRZ bit enables the DSPI transfers to be stopped on the next frame boundary when the SoC enters Debug mode. 0 Do not stop serial transfers 1 Stop serial transfers
MTFE	Modified Timing Format Enable. The MTFE bit enables a modified transfer format to be used. See *Section 28.4.6.4: Modified SPI/DSI 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. Refer to *Section 28.2.2.4: Peripheral Chip Select 5 / Peripheral Chip Select Strobe (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. The ROOE bit enables in 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, generated the overflow, is ignored or shifted in to the shift register. See *Section 28.4.11.6: Receive FIFO Overflow Interrupt Request*, for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
PCSIS*x*	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. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See *Section 28.4.12: Power Saving Features*, for more information. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
DIS_TXF	Disable Transmit FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See *Section 28.4.2.3: FIFO Disable Operation*, for details. 0 TX FIFO is enabled 1 TX FIFO is disabled

Table 325. DSPI_MCR Field Descriptions (continued)

Field	Description
DIS_RXF	Disable Receive FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See *Section 28.4.2.3: FIFO Disable Operation*, for details. 0 RX FIFO is enabled 1 RX FIFO is disabled
CLR_TXF	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
CLR_RXF	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
SMPL_PT	Sample Point. SMPL_PT field controls when the DSPI master samples SIN in Modified Transfer Format. *Figure 396* shows where the master can sample the SIN pin. 00 DSPI samples SIN at driving SCK edge. 01 DSPI samples SIN one system clock after driving SCK edge 10 DSPI samples SIN two system clocks after driving SCK edge 11 Reserved
PES	Parity Error Stop. PES bit controls SPI operation when a parity error detected in received SPI frame. 0 SPI frames transmission continue. 1 SPI frames transmission stop.
HALT	Halt. The HALT bit starts and stops DSPI transfers. See *Section 28.4.1: Start and Stop of DSPI Transfers*, for details on the operation of this bit. 0 Start transfers 1 Stop transfers

1. If the FIFO is enabled with continuous SCK mode, before setting the CONT_SCKE bit, the TX-FIFO should be cleared and only DSPI_CTAR0 register should be used for transfer attributes. Otherwise a change in SCK frequency occurs.

28.3.2.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter, that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write the DSPI_TCR, when the DSPI is in the Running state.

Offset: 0x8 Access: Read/Write

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

Figure 368. DSPI Transfer Count Register (DSPI_TCR)

Table 326. DSPI_TCR Field Descriptions

Field	Description
TCNT	SPI Transfer Counter. The TCNT field counts the number of SPI transfers the DSPI makes. The TCNT field increments every time the last bit of a SPI frame is transmitted. A value written to TCNT presets the counter to that value. 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' i.e. incrementing the counter past 65535 resets the counter to zero.

28.3.2.3 DSPI Clock and Transfer Attributes Registers 0–5 (DSPI_CTAR0–DSPI_CTAR5)

The DSPI_CTAR registers are used to define different transfer attributes. Do not write to the DSPI_CTAR registers, while the DSPI is in the Running state.

In master mode, the DSPI_CTAR0 - DSPI_CTAR5 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 bitfields in the DSPI_CTAR0 and DSPI_CTAR1 registers are used to set the slave transfer attributes.

When the DSPI is configured as a SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI_CTAR register is used. When the DSPI is configured as a SPI bus slave, the DSPI_CTAR0 register is used.

When the DSPI is configured as a DSI master, the DSICTAS field in the DSPI DSI Configuration Register (DSPI_DSICR), selects which of the DSPI_CTAR register is used. When the DSPI is configured as a DSI bus slave, the DSPI_CTAR1 register is used.

In CSI Configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI Configuration follow the protocol described for SPI Configuration, and DSI transfers in CSI Configuration follow the protocol described for DSI Configuration. CSI Configuration is only valid in conjunction with master mode. See *Section 28.4.4: Combined Serial Interface (CSI) Configuration*, for more details.

TSB mode sets some limitations on transfer attributes:

- Clock phase is forced to be CPHA = 1 and the CPHA bit setting has no effect.
- PCS lines are driven at the driving edge of the SCK clock together with SOUT, so PCS assertion and negation delays control is unavailable and PCSSCK, PASC, CSSCK and ASC fields have no effect.
- Delay after transfer can be set from 1 to 64 serial clocks with help of PDT and DT fields.

Offset: 0x000C (DSPI*x*_CTAR0)
Offset: 0x0010 (DSPI*x*_CTAR1)
Offset: 0x0014 (DSPI*x*_CTAR2)
Offset: 0x0018 (DSPI*x*_CTAR3)
Offset: 0x001C (DSPI*x*_CTAR4)
Offset: 0x0020 (DSPI*x*_CTAR5)

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

Figure 369. DSPI Clock and Transfer Attributes Register 0–5 (DSPI_CTAR0–DSPI_CTAR5) in the master mode

Offset: 0x000C (DSPI*x*_CTAR0)
Offset: 0x0010 (DSPI*x*_CTAR1)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FMSZ					CPOL	CPHA	PE	PP	Not used						
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	Not used															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 370. DSPI Clock and Transfer Attributes Register 0, 1 (DSPI_CTAR0, DSPI_CTAR1) in the slave mode

Table 327. DSPI_CTAR*n* Field Descriptions in master mode

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 328*. See the BR 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 number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3. When operating in TSB mode, detailed in *Section 28.4.9: Timed Serial Bus (TSB)*, the FMSZ field value plus 1 is equal the data frame bit number, where control of the PCS assertion switches from the DSPI_DSICR to the DSPI_DSICR1 register.
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. In Continuous SCK mode or TSB mode the bit value is ignored and the transfers are done as CPHA bit is set to 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
LSBFE	LSB First. The LSBFE bit selects if the LSB or MSB of the frame is transferred first. 0 Data is transferred MSB first 1 Data is transferred LSB first
PCSSCK	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. See the CSSCK field description how to compute the PCS to SCK Delay. In the TSB mode the PCSSCK field has no effect. 00 PCS to SCK Prescaler value is 1 01 PCS to SCK Prescaler value is 3 10 PCS to SCK Prescaler value is 5 11 PCS to SCK Prescaler value is 7
PASC	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. See the ASC field description how to compute the After SCK Delay. In the TSB mode the PASC field has no effect. 00 After SCK Delay Prescaler value is 1 01 After SCK Delay Prescaler value is 3 10 After SCK Delay Prescaler value is 5 11 After SCK Delay Prescaler value is 7

Table 327. DSPI_CTAR*n* Field Descriptions in master mode

Field	Descriptions
PDT	Delay after Transfer Prescaler. The PDT field selects the prescaler value for the delay between the negation of the CS*x* 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. In the TSB mode the PDT field defines two MSB bits of the Delay after Transfer. See the DT field description for details on how to compute the Delay after Transfer. 00 Delay after Transfer Prescaler value is 1 01 Delay after Transfer Prescaler value is 3 10 Delay after Transfer Prescaler value is 5 11 Delay after Transfer Prescaler value is 7
PBR	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. See the BR field description for details on how to compute the baud rate. 00 Baud Rate Prescaler value is 2 01 Baud Rate Prescaler value is 3 10 Baud Rate Prescaler value is 5 11 Baud Rate Prescaler value is 7
CSSCK	CS to SCK Delay Scaler. The CSSCK field selects the scaler value for the CS to SCK delay. This field is only used in master mode. The CS to SCK Delay is the delay between the assertion of PCS and the first edge of the SCK. *Table 329* list the scaler values.The CS to SCK Delay is a multiple of the system clock period and it is computed according to the following equation: **Equation 10** $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$ See *Section 28.4.5.2: CS to SCK delay (tCSC)*, for more details. In the TSB mode the field has no effect.
ASC	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 329* list 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 11** $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$ See *Section 28.4.5.3: After SCK Delay (tASC)*, for more details. In the TSB mode the field has no effect.

Table 327. DSPI_CTAR*n* Field Descriptions in master mode

Field	Descriptions
DT	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 CS*x* signal at the end of a frame and the assertion of PCS at the beginning of the next frame. *Table 329* lists the scaler values. In the Continuous Serial Communications Clock operation the DT value is fixed to one SCK clock period, The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation: **Equation 12** $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ In the TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See *Section 28.4.5.4: Delay after Transfer (tDT)*, for more details.
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 330* lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: **Equation 13** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section 28.4.5.1: Baud Rate Generator*, for more details.

Table 328. 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 329. Delay Scaler Encoding

Field value	Scaler Value	Field value	Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096

Table 329. Delay Scaler Encoding (continued) (continued)

Field value	Scaler Value	Field value	Scaler Value
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 330. DSPI Baud Rate Scaler

BR	Baud Rate Scaler Value	BR	Baud Rate Scaler Value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

Table 331. DSPI_CTAR*0*, DSPI_CTAR*1* Field Descriptions in slave mode

Field	Descriptions
FMSZ	Frame Size. The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3.
CPOL	Clock Polarity. The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). 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. 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
PE	Parity Enable. PE bit enables parity bit transmission and reception for the frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
PP	Parity Polarity. PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.

28.3.2.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a ‘1’ to it. 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.

Offset: 0x002C Access: Read

	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	DPEF	SPEF	DDIF	RFOF	0	RFDF	0
W	w1c	w1c		w1c	w1c		w1c			w1c	w1c	w1c	w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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

Figure 371. DSPI Status Register (DSPI_SR)

Table 332. DSPI_SR Field Descriptions

Field	Description
TCF	Transfer Complete Flag. The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit remains set until cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete
TXRXS	TX & RX Status. The TXRXS bit reflects the run status of the DSPI. See *Section 28.4.1: Start and Stop of DSPI Transfers*, what causes this bit to be set or cleared. 0 TX and RX operations are disabled (DSPI is in STOPPED state) 1 TX and RX operations are enabled (DSPI is in RUNNING state)
EOQF	End of Queue Flag. The EOQF bit indicates that the last entry in a queue has been transmitted when the DSPI in the master mode. 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. The EOQF bit remains set until cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executed command 1 EOQ bit is set in the executed SPI command
TFUF	Transmit FIFO Underflow Flag. The TFUF bit indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit remains set until cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
TFFF	Transmit FIFO Fill Flag. The TFFF bit 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 by acknowledgement from the DMA controller to the TX FIFO full request. 0 TX FIFO is full 1 TX FIFO is not full
DPEF	DSI Parity Error Flag. The DPEF flag indicates that a DSI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred 1 Parity Error has occurred

Table 332. DSPI_SR Field Descriptions (continued)

Field	Description
SPEF	SPI Parity Error Flag. The SPEF flag indicates that a SPI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred 1 Parity Error has occurred
DDIF	DSI data received with active bits. The DDIF flag indicates that DSI frame had been received with bits, selected by DSPI_DIMR with active polarity, defined by DSPI_DPIR register. The bit remains set until cleared by writing 1 to it. 0 No DSI data with active bits was received 1 DSI data with active bits was received
RFOF	Receive FIFO Overflow Flag. The RFOF bit indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit remains set until cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
RFDF	Receive FIFO Drain Flag. The RFDF bit 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 DMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty
TXCTR	TX FIFO Counter. The TXCTR field 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 a SPI command is executed and the SPI data is transferred to the shift register.
TXNXTPTR	Transmit Next Pointer. The TXNXTPTR field 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 28.4.11.4: Transmit FIFO Underflow Interrupt Request*, for more details.
RXCTR	RX FIFO Counter. The RXCTR field indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented every time data is transferred from the shift register to the RX FIFO.
POPNXTPTR	Pop Next Pointer. The POPNXTPTR field contains a pointer to the RX FIFO entry that will be returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See *Section 28.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism*, for more details.

28.3.2.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER register controls DMA and interrupt requests. Do not write to the DSPI_RSER while the DSPI is in the Running state.

Offset: 0x0030 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCFRE	0	0	EOQFRE	TFUFRE	0	TFFFRE	TFFFDIRS	0	DPEFRE	SPEFRE	DDIFRE	RFOFRE	0	RFDFRE	RFDFDIRS
W																
Reset	0	0	0	0	0	0	0	0	0	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

Figure 372. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 333. DSPI_RSER Field Descriptions

Field	Description
TCFRE	Transmission Complete Request Enable. The TCFRE bit enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
EOQFRE	DSPI Finished Request Enable. The EOQFRE bit enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
TFUFRE	Transmit FIFO Underflow Request Enable. The TFUFRE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
TFFFRE	Transmit FIFO Fill Request Enable. The TFFFRE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFFDIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
TFFFDIRS	Transmit FIFO Fill DMA or Interrupt Request Select. The TFFFDIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the DSPI_RSER[TFFFRE] bit is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is generated 1 DMA request is generated
DPEFRE	DSI Parity Error Request Enable. The DPEFRE bit enables DPEF flag in the DSPI_SR to generate an interrupt requests. 0 DPEF interrupt requests are disabled 1 DPEF interrupt requests are enabled
SPEFRE	SPI Parity Error Request Enable. The SPEFRE bit enables SPEF flag in the DSPI_SR to generate an interrupt requests. 0 SPEF interrupt requests are disabled 1 SPEF interrupt requests are enabled

Table 333. DSPI_RSER Field Descriptions (continued)

Field	Description
DDIFRE	DSI data received with active bits Request Enable. The DDIFRE bit enables the DDIF flag in the DSPI_SR to generate an interrupt requests. 0 DDIF interrupt requests are disabled 1 DDIF interrupt requests are enabled
RFOFRE	Receive FIFO Overflow Request Enable. The RFOFRE bit enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
RFDFRE	Receive FIFO Drain Request Enable. The RFDFRE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDFDIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
RFDFDIRS	Receive FIFO Drain DMA or Interrupt Request Select. The RFDFDIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the DSPI_RSER[RFDFRE] bit register is set, the RFDFDIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated

28.3.2.6 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR register provides means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See *Section 28.4.2.4: Transmit First In First Out (TX FIFO) Buffering Mechanism* for more information. Eight or sixteen bit write accesses to the DSPI_PUSHR transfers all 32 register bits to the TX FIFO.The register structure is different in master and slave modes. In master mode the register provides 16-bit command and 16-bit data to the TX FIFO. In slave mode all 32 register bits can be used as data, supporting up to 32-bit SPI frame operation.

Offset: 0x0034 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CONT	CTAS			EOQ	CTCNT	PE	PP	0	0	PCS5 (1)	PCS4 (1)	PCS3 (1)	PCS2 (1)	PCS1 (1)	PCS0 (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 W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 373. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode

1. Please refer to the *Table 323*.

Table 334. DSPI_PUSHR Field Descriptions in master mode

Field	Descriptions
CONT	Continuous Peripheral Chip Select Enable. The CONT bit selects a Continuous Selection Format. The bit is used in SPI master mode. The bit enables the selected CS*x* signals to remain asserted between transfers. See *Section 28.4.6.5: 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
CTAS	Clock and Transfer Attributes Select. The CTAS field selects number of the DSPI_CTAR register be used to set the transfer attributes for the associated SPI frame. The field is only used in SPI master mode. In SPI slave mode DSPI_CTAR0 is used. The number of DSPI_CTAR registers is implementation specific and the CTAS should be set to select only implemented one.
EOQ	End Of Queue. The EOQ bit provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI_SR is set. 0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer
CTCNT	Clear Transfer Counter. The CTCNT bit clears the DSPI_TCR[TCNT] field. The TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear DSPI_TCR[TCNT] 1 Clear DSPI_TCR[TCNT]
PE	Parity Enable. PE bit enables parity bit transmission and parity reception check for the SPI frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
PP	Parity Polarity. PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
PCS*x*	Peripheral chip select *0* 0–5. Selects which CS*x* signals are asserted for the transfer. 0 Negate the CS*x* signal 1 Assert the CS*x* signal
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred according to the associated SPI command.

Offset: 0x00x34 Access: Read/Write

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

Figure 374. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode

Table 335. DSPI_PUSHR Field Descriptions in slave mode

Field	Descriptions
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred.

28.3.2.7 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides means to read the RX FIFO. See *Section 28.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism* for a description of the RX FIFO operations. Eight or sixteen bit read accesses to the DSPI_POPR have the same effect on the RX FIFO as 32-bit read access.

Offset: 0x0038 Access: Read

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

Figure 375. DSPI POP RX FIFO Register (DSPI_POPR)

Table 336. DSPI_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.

28.3.2.8 DSPI Transmit FIFO Registers 0–3 (DSPI_TXFR0–DSPI_TXFR3)

The DSPI_TXFR0 - DSPI_TXFR3 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_TXFRx registers does not alter the state of the TX FIFO. If a four entry TX FIFO is implemented, DSPI_TXFR0 - DSPI_TXFR3 are accessible.

Offset: 0x003C (DSPI*x*_TXFR0) Access: Read
Offset: 0x0040 (DSPI*x*_TXFR1)
Offset: 0x0044 (DSPI*x*_TXFR2)
Offset: 0x0048 (DSPI*x*_TXFR3)

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

Figure 376. DSPI Transmit FIFO Register 0–3 (DSPI_TXFR0–DSPI_TXFR3)

Table 337. DSPI_TXFR*n* Field Descriptions

Field	Description
TXCMD/ TXDATA[0:15]	Transmit Command or Transmit Data. In master mode the TXCMD field contains the command that sets the transfer attributes for the SPI data. See *Section 28.3.2.6: DSPI PUSH TX FIFO Register (DSPI_PUSHR)*, for details on the command field. In slave mode the TXDATA contains 16 MSB bits of the SPI data to be shifted out
TXDATA[16:31]	Transmit Data. The TXDATA field contains the SPI data to be shifted out.

28.3.2.9 DSPI Receive FIFO Registers 0–3 (DSPI_RXFR0–DSPI_RXFR3)

The DSPI_RXFR0 - DSPI_RXFR3 registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFRx registers does not alter the state of the RX FIFO. If a four entry RX FIFO is implemented, DSPI_RXFR0 - DSPI_RXFR3 exist, for example.

Offset: 0x007C (DSPI*x*_RXFR0) Access: Read
Offset: 0x0080 (DSPI*x*_RXFR1)
Offset: 0x0084 (DSPI*x*_RXFR2)
Offset: 0x0088 (DSPI*x*_RXFR3

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

Figure 377. DSPI Receive FIFO Registers 0–3 (DSPI_RXFR0–DSPI_RXFR3)

Table 338. DSPI_RXFR*n* Field Descriptions

Field	Description
RXDATA	Receive Data. The RXDATA field contains the received SPI data.

28.3.2.10 DSPI DSI Configuration Register (DSPI_DSICR)

The DSI Configuration Register selects various attributes associated with DSI and CSI Configurations. Do not write to the DSPI_DSICR, while the DSPI is in the Running state.

Offset: 0x00BC Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTOE	FMSZ4	MTOCNT						0	0	0	TSBC	TXSS	TPOL	TRRE	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCONT	DSICTAS			DMS	PES	PE	PP	0	0	DPCS5(1)	DPCS4(1)	DPCS3(1)	DPCS2(1)	DPCS1(1)	DPCS0(1)
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 378. DSPI DSI Configuration Register (DSPI_DSICR)

1. Please refer to the *Table 339*.

Table 339. DSPI_DSICR Field Descriptions

Field	Description
MTOE	Multiple Transfer Operation Enable. The MTOE bit enables multiple DSPIs to be connected in a parallel or serial configuration. See *Section 28.4.3.6: Multiple Transfer Operation (MTO)*, for more information. 0 Multiple Transfer Operation disabled 1 Multiple Transfer Operation enabled The MTOE and TSB bits should not be set simultaneously.
FMSZ4	MSB of the Frame Size in master mode. if the bit is set, 16 is added to the frame size, defined by DSPI_CTARn[FMSZ] field. DSPI_CTARn register is selected by the DSICTAS field.
MTOCNT	Multiple Transfer Operation Count. The MTOCNT field selects number of bits to be shifted out during a transfer in Multiple Transfer Operation. The field sets the number of SCK cycles that the bus master will generate to complete the transfer. The number of SCK cycles used will be one more than the value in the MTOCNT field. The number of SCK cycles defined by MTOCNT must be equal to or greater than the frame size. When TSBC is set, MTOCNT field has no effect.
TSBC	Timed Serial Bus Configuration. The TSBC bit enables the Timed Serial Bus Configuration. This configuration allows 32-bit data to be used. It also allows t_{DT} to be programmable. See *Section 28.4.9: Timed Serial Bus (TSB)* for detailed information. 0 Timed Serial Bus Configuration disabled 1 Timed Serial Bus Configuration enabled If this bit is clear the DSPI_DSICR1 register value has no effect.

Table 339. DSPI_DSICR Field Descriptions (continued)

Field	Description
TXSS	Transmit Data Source Select. The TXSS bit selects the source of data to be serialized. The source can be either data from host Software written to the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), or Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR). 0 Source of serialized data is the DSPI_SDR 1 Source of serialized data is the DSPI_ASDR
TPOL	Trigger Polarity. The TPOL bit selects the active edge of the hardware trigger input signal (HT). initiating DSI frames transfer. See *Section 28.4.3.4: DSI Deserialization*, for more information. 0 Falling edge will initiate a transfer 1 Rising edge will initiate a transfer
TRRE	Trigger Reception Enable. The TRRE bit enables the DSPI to initiate DSI frames transfer with external trigger signal. See *Section 28.4.3.4: DSI Deserialization*, for more information. 0 Trigger signal reception disabled 1 Trigger signal reception enabled
CID	Change In Data Transfer Enable. The CID bit enables a change in serialization data to initiate DSI frames transfer. in DSI and CSI configurations. When the CID bit is set, DSI frames are initiated when the current DSI data differs from the previous DSI data shifted out. Refer to *Section 28.4.3.4: DSI Deserialization*, for more information.
DCONT	DSI Continuous Peripheral Chip Select Enable. The DCONT bit enables the CS*x* signals to remain asserted between transfers. The DCONT bit only affects the CS*x* signals in DSI master mode. See *Section 28.4.6.5: Continuous Selection Format*, for details. When TSBC bit is set, DCONT bit has no effect. 0 Return Peripheral Chip Select signals to their inactive state after transfer is complete 1 Keep Peripheral Chip Select signals asserted after transfer is complete
DSICTAS	DSI Clock and Transfer Attributes Select. The DSICTAS field selects which of the DSPI_CTAR register is used to provide transfer attributes for DSI frames. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPI_CTAR1 is always selected.
DMS	Data Match Stop. DMS bit if set stops DSI frames transmissions if DDIF flag is set in the DSPI_SR register. 0 DDIF flag does not have effect on DSI frames transmissions. 1 DDIF flag stops DSI frame transmissions.
PES	Parity Error Stop. PES bit if set stops DSI operation if the parity error had happened in received DSI frame. 0 parity error does not stop DSI frame transmissions. 1 parity error stops all DSI frame transmissions.
PE	Parity Enable. PE bit enables parity bit transmission and parity reception check for the DSI frames 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
PP	Parity Polarity. PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[DPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[DPEF] bit is set if in the received frame number of “1” bits is even.
DPCS*x*	DSI Peripheral Chip Select 0–5. The DPCS bits select which of the CS*x* signals to assert during a DSI master mode transfer. 0 Negate CS*x* 1 Assert CS*x*

28.3.2.11 DSPI DSI Serialization Data Register (DSPI_SDR)

The DSPI_SDR contains the states of the Parallel Input signals. The states of the Parallel Input signals are latched into the DSPI_SDR on the rising edge of every system clock. The DSPI_SDR is read-only. When the TXSS bit in the DSPI_DSICR is cleared, the data in the DSPI_SDR is used as the source of the DSI frames.

Offset: 0x00C0 Access: Read

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

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

Figure 379. DSPI DSI Serialization Data Register (DSPI_SDR)

Table 340. DSPI_SDR Field Descriptions

Field	Description
SER_DATA	Serialized Data. The SER_DATA field contains the signal states of the Parallel Input signals.

28.3.2.12 DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

The DSPI_ASDR provides means for host software to write the data to be serialized. When the TXSS bit in the DSPI_DSICR is set, the data in the DSPI_ASDR is the source of the DSI frames. Writes to the DSPI_ASDR take effect on the next frame boundary.

Offset: 0x00C4 Access: Read/Write

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

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

Figure 380. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

Table 341. DSPI_ASDR Field Descriptions

Field	Descriptions
ASER_DATA	Alternate Serialized Data. The ASER_DATA field holds the alternate data to be serialized.

28.3.2.13 DSPI DSI Transmit Comparison Register (DSPI_COMPR)

The DSPI_COMPR holds a copy of the last transmitted DSI data. The DSPI_COMPR is read-only. DSI data is transferred to this register as it is loaded into the TX Shift Register.

Offset: 0x00C8 Access: Read/Write

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

Figure 381. DSPI DSI Transmit Comparison Register (DSPI_COMPR)

Table 342. DSPI_COMPR Field Descriptions

Field	Description
COMP_DATA	Compare Data. The COMP_DATA field holds the last serialized DSI data.

28.3.2.14 DSPI DSI Deserialization Data Register (DSPI_DDR)

The DSPI_DDR register holds the signal states for the Parallel Output signals. The DSPI_DDR is read-only and host software can read data from incoming DSI frames.

Offset: 0x00CC Access: Read

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

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

Figure 382. DSPI Deserialization Data Register (DSPI_DDR)

Table 343. DSPI_DDR Field Descriptions

Field	Descriptions
DESER_DATA	Deserialized Data. The DESER_DATA field holds deserialized data which is presented as signal states to the Parallel Output signals.

28.3.2.15 DSPI DSI Configuration Register 1 (DSPI_DSICR1)

The DSI Configuration Register 1 selects various attributes associated with TSB Configuration. The user must not write to the DSPI_DSICR1 while the DSPI is in the Running state. If TSBC bit is cleared the register value is ignored.

Offset: 0x00D0 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	TSBCNT					0	0	0	0	0	0	DSE1	DSE0
W																
Reset	0	0	0	0	0	0	0	0	0	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	DPCS1_5(1)	DPCS1_4(1)	DPCS1_3(1)	DPCS1_2(1)	DPCS1_1(1)	DPCS1_0(1)
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 383. DSPI DSI Configuration Register 1 (DSPI_DSICR1)

1. Please refer to the *Table 339*.

Table 344. DSPI_DSICR1 Field Descriptions

Field	Description
TSBCNT	Timed Serial Bus Operation Count. When TSBC is set, TSBCNT defines the length of the data frame. TSBCNT field valid value is from 3 to 31. The TSBCNT field selects number of data bits to be shifted out during a transfer in TSB mode. The number of data bits in the data frame is one more than the value in the TSBCNT field.
DSE1	Data Select Enable1. When TBSC bit is set, the DSE1 bit controls insertion of the zero bit (Data Select) in the middle of the data frame. The insertion bit position is defined by FMSZ field of DSPI_CTARn register, selected by the DSPI_DSICR[DSICTAS] field. The TSBCNT field value must be greater than the FMSZ field value plus one for proper operation of the DSE1 bit. 1 Zero bit is inserted at the middle of the data frame. Total number of bits in the data frame is increased by 1. 0 No Zero bit inserted in the middle of the data frame.
DSE0	Data Select Enable0. The DSE0 bit controls insertion of the zero bit (Data Select) in the beginning of the DSI frame. 1 Zero bit is inserted at the beginning of the data frame. Total number of bits in the data frame is increased by 1. 0 No Zero bit inserted in the beginning of the frame
DPCS1_*x*	DSI Peripheral Chip Select 0–5. These bits define the PCSs to assert for the second part of the DSI frame when operating in TSB configuration with dual receiver. The DPCS1 bits select which of the CS*x* signals to assert during the second part of the DSI frame. The DPCS1 bits only control the assertions of the CS*x* signals in TSB mode. 0 Negate the CS*x* signal 1 Assert the CS*x* signal

28.3.2.16 DSPI DSI Serialization Source Select Register (DSPI_SSR)

DSPI DSI Serialization Source Select Register provides means to create combined frame for transmission, containing bits from DSPI_ASDR register and from DSPI_SDR register. Each bit in the DSPI_SSR register selects corresponding bit to be serialized. When DSPI_DSICR[TXSS] is set, the DSPI_SSR register value has no effect.

Offset: 0x00D4 Access: Read/Write

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

Figure 384. DSPI DSI Serialization Source Select Register (DSPI_SSR)

Table 345. DSPI_SSR Field Descriptions

Field	Description
SS	Source Select. The SS bits select serialization source for DSI frame. Each SS bit selects data for corresponded bit in the transmitted frame. 0 the bit in transmitted frame is taken from Parallel Input pin; 1 the bit in transmitted frame is taken from DSPI_ASDR register

28.3.2.17 DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)

The DSPI DSI Deserialized Data Interrupt Mask Register selects bits in the received DSI frame to be checked to generate the DDI interrupt.

Offset: 0x00E8 Access: Read/Write

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

Figure 385. DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)

Table 346. DSPI_DIMR Field Descriptions

Field	Description
MASK	MASK. The MASK bits define which bits in received deserialization data should be checked to produce the Deserialized Data Interrupt (DDI). 0 the bit in received DSI frame does not produce DDI interrupt. 1 the bit in received DSI frame can produce DDI interrupt if the data bit matches to configured polarity.

28.3.2.18 DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)

The DSPI DSI Deserialized Data Polarity Interrupt Register defines what data bits value in the received DSI frame generates the DDI interrupt.

Offset: 0x00EC Access: Read/Write

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

Figure 386. DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)

Table 347. DSPI_DPIR Field Descriptions

Field	Description
DP	Data Polarity. The DP bits define what value of the received deserialization data sets the DSPI_SR[DDIF] bit. 0 if received bit is 0 the DSPI_SR[DDIF] bit is set. 1 if received bit is 1 the DSPI_SR[DDIF] bit is set.

28.4 Functional Description

The Deserial Serial Peripheral Interface (DSPI) block supports full-duplex, synchronous serial communications between MCUs and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing up to 32 Parallel Input/Output signals. All communications are done with SPI-like protocol.

The DSPI has three configurations:

- SPI Configuration in which the DSPI operates as a basic SPI or a queued SPI.
- DSI Configuration in which the DSPI serializes Parallel Input/Output signals or bits from memory mapped register.
- CSI Configuration in which the DSPI combines the functionality of the SPI and DSI configurations.

The DCONF field in the *Section 28.3.2.1: DSPI Module Configuration Register (DSPI_MCR)* determines the DSPI Configuration. See *Table 325* for the DSPI configuration values.

The DSPI_CTAR0 - DSPI_CTAR5 registers hold clock and transfer attributes. The SPI configuration allows to select which CTAR to use on a frame by frame basis by setting a field in the SPI command. The DSI configuration statically selects which CTAR to use. In CSI Configuration priority logic determines if SPI data or DSI data is transferred and dictates what CTAR register is used for the data transfer. See *Section 28.3.2.3: DSPI Clock and Transfer Attributes Registers 0–5 (DSPI_CTAR0–DSPI_CTAR5)*, for information on the fields of the DSPI_CTAR registers.

Typical master to slave connections are shown in the *Figure 387*. When a data transfer operation is performed, data is serially shifted a predetermined number of bit positions. Because the modules are linked, data is exchanged between the master and the slave. The data that was in the master 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_SR is set to indicate a completed transfer.

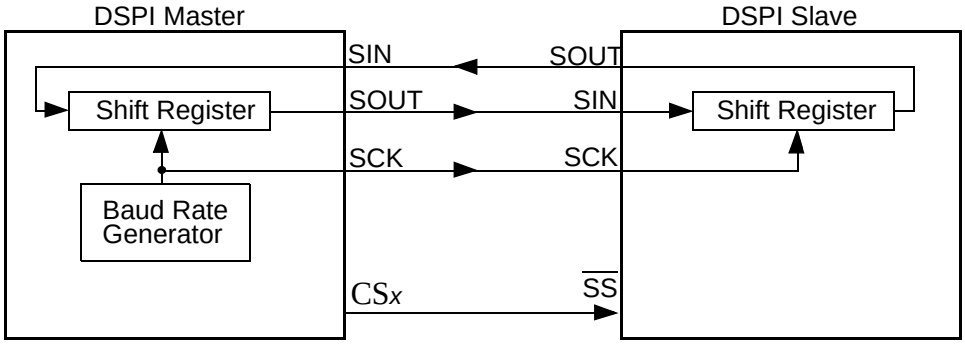


Figure 387. SPI and DSI Serial Protocol Overview

Generally more than one slave device can be connected to the DSPI master. Eight Peripheral Chip Select (CS*x*) signals of the DSPI masters can be used to select which of the slaves to communicate with.

The three DSPI configurations share transfer protocol and timing properties which are described independently of the configuration in *Section 28.4.6: Transfer Formats*. The transfer rate and delay settings are described in *Section 28.4.5: DSPI Baud Rate and Clock Delay Generation*.

28.4.1 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. In the RUNNING state serial transfers take place.

The TXRXS bit in the DSPI_SR indicates in what state the DSPI is. The bit is set if the module in RUNNING state.

The DSPI is started (DSPI transitions to RUNNING) when all of the following conditions are true:

- DSPI_SR[EOQF] bit is clear
- SoC is not in the debug mode is or the DSPI_MCR[FRZ] bit is clear
- DSPI_MCR[HALT] bit is clear

The DSPI stops (transitions from RUNNING to STOPPED) after the current frame when any one of the following conditions exist:

- DSPI_SR[EOQF] bit is set
- SoC in the debug mode and the DSPI_MCR[FRZ] bit is set
- DSPI_MCR[HALT] bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or immediately if no transfers are in progress.

28.4.2 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_MCR is 0b00. The SPI frames can be from four to sixteen bits long. Host CPU or a DMA controller transfer the SPI data from the external to DSPI RAM queues to a transmit First-In First-Out (TX FIFO) buffer. The received data is stored in entries in the Receive FIFO (RX FIFO) buffer. Host CPU or the DMA controller transfer the received data from the RX FIFO to memory external to the DSPI. The FIFO buffers operation is described in *Section 28.4.2.4: Transmit First In First Out (TX FIFO) Buffering Mechanism*, and *Section 28.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism*. The interrupt and DMA request conditions are described in *Section 28.4.11: Interrupts/DMA Requests*.

The SPI Configuration supports two block-specific modes - master mode and slave mode. The FIFO operations are similar for both modes. 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 space is used for 16 most significant bit of the transmit data.

28.4.2.1 Master Mode

In SPI master mode the DSPI initiates the serial transfers by controlling the Serial Communications Clock (SCK) and the Peripheral Chip Select (CS*x*) signals. The SPI command field in the executing TX FIFO entry determines which CTAR registers will be 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. See *Section 28.3.2.6: DSPI PUSH TX FIFO Register (DSPI_PUSHR)* for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the Serial Out (SOUT) 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.

28.4.2.2 Slave Mode

In SPI slave mode the DSPI responds to transfers initiated by a 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 a SPI master. The SPI slave mode transfer attributes are set in the DSPI_CTAR0. The data is shifted out with MSB first.

28.4.2.3 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 FIFOs are disabled separately; setting the DSPI_MCR[DIS_TXF] bit disables the TX FIFO, and setting the DSPI_MCR[DIS_RXF] bit disables the RX FIFO.

The FIFO Disable mechanisms are transparent to the user and to host software; Transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR.When the TX FIFO is disabled the TFFF, TFUF and TXCTR fields in DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_TXFR registers and TXNXTPTR are undefined. Likewise, when the RX FIFO is disabled, the RFDF, RFOF and RXCTR fields in the DSPI_SR behave as if there is a one-entry FIFO, but the contents of the DSPI_RXFR registers and POPNXTPTR are undefined.

28.4.2.4 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 from one to sixteen words, each consisting of a command field and a data field. The number of entries in the TX FIFO is SoC specific.SPI commands and data are added to the TX FIFO by writing to the *Section 28.3.2.6: DSPI PUSH TX FIFO Register (DSPI_PUSHR)*. TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO Counter field (TXCTR) in the *Section 28.3.2.4: DSPI Status Register (DSPI_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.

The TXNXTPTR field indicates which TX FIFO Entry will be transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI_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. The maximum value of the field is equal to DSPI_HCR[TXFR] and it rolls over after reaching the maximum.

28.4.2.4.1 Filling the TX FIFO

Host software or other intelligent blocks can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR. When the TX FIFO is not full, the TX FIFO Fill Flag (TFFF) in the DSPI_SR is set. The TFFF bit is cleared when TX FIFO is full and the DMA controller indicates that a write to DSPI_PUSHR is complete. Writing a ‘1’ to the TFFF bit also clears it. The TFFF can generate a DMA request or an interrupt request. See *Section 28.4.11.2: Transmit FIFO Fill Interrupt or DMA Request*, for details.

The DSPI ignores attempts to push data to a full TX FIFO, the state of the TX FIFO does not change and no error condition is indicated.

28.4.2.4.2 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 decrements by one. At the end of a transfer, the TCF bit in the DSPI_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_MCR.

If an external 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_SR is set. See *Section 28.4.11.4: Transmit FIFO Underflow Interrupt Request*, for details.

28.4.2.5 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 from one to sixteen received SPI data frames. The number of entries in the RX FIFO is SoC specific.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 are removed (popped) from the RX FIFO by reading the *Section 28.3.2.7: DSPI POP RX FIFO Register (DSPI_POPR)*. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO.

The RX FIFO Counter field (RXCTR) in the *Section 28.3.2.4: DSPI Status Register (DSPI_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_SR points to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR contains the positive offset from DSPI_RXFR0 in number of 32-bit registers. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that will be returned when DSPI_POPR is read. The POPNXTPTR field is incremented every time the DSPI_POPR is read. The maximum value of the field is equal to DSPI_HCR[RXFR] and it rolls over after reaching the maximum.

28.4.2.5.1 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 a 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_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI_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 cleared, the incoming data is ignored.

28.4.2.5.2 Draining the RX FIFO

Host CPU or a DMA can remove (pop) entries from the RX FIFO by reading the *Section 28.3.2.7: DSPI POP RX FIFO Register (DSPI_POPR)*. A read of the DSPI_POPR decrements the RX FIFO Counter by one. Attempts to pop data from an empty RX FIFO are ignored and the RX FIFO Counter remains unchanged. The data, read from the empty RX FIFO, is undetermined.

When the RX FIFO is not empty, the RX FIFO Drain Flag (RFDF) in the DSPI_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the DMA controller indicates that a read from DSPI_POPR is complete or by writing a '1' to it.

28.4.3 Deserial Serial Interface (DSI) Configuration

The DSI Configuration supports pin count reduction by serializing Parallel Input signals or register bits and shifting them out in a SPI-like protocol. The timing and transfer protocol is described in *Section 28.4.6: Transfer Formats*. The various features of the DSI Configuration are set in *Section 28.3.2.10: DSPI DSI Configuration Register (DSPI_DSICR)*.

The DSI frames can be from four to 32 bits. With Multiple Transfer Operation (MTO) the DSPI supports serial chaining of DSPI blocks within an SoC to create DSI frames up to 64 bits, consisting of concatenated bits from multiple DSPIs. The DSPI also supports parallel chaining allowing several DSPIs and off-chip SPI devices to share the same Serial Communications Clock (SCK) and Peripheral Chip Select (CS*x*) signals. See *Section 28.4.3.6: Multiple Transfer Operation (MTO)*, for details on the serial and parallel chaining support.

28.4.3.1 DSI Master Mode

In DSI master mode the DSPI initiates and controls the DSI transfers. The DSI master has four different conditions that can initiate a transfer:

- Continuous
- Change in data
- Trigger signal
- Trigger signal combined with a change in data

The four transfer initiation conditions are described in *Section 28.4.3.4: DSI Deserialization*. Transfer attributes are set during initialization. The DSICTAS field in the DSPI_DSICR determines which of the DSPI_CTAR registers will control the transfer attributes.

28.4.3.2 Slave Mode

In DSI slave mode the DSPI responds to transfers initiated by a SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode Transfer attributes are set in the DSPI_CTAR1. The data is shifted out with MSB first.

If the CID bit in the DSPI_DSICR is set and the data in the DSPI_COMPR differs from the selected source of the serialized data, the slave DSPI will assert the MTRIG signal. If the slave's HT signal is asserted and the TRRE is set, the slave DSPI asserts MTRIG. These features are included to support chaining of several DSPI. Details about the MTRIG signal is found in *Section 28.4.3.6: Multiple Transfer Operation (MTO)*.

28.4.3.3 DSI Serialization

In the DSI Configuration from four to sixteen bits can be serialized using two different sources. The TXSS bit in the DSPI_DSICR selects between the *Section 28.3.2.11: DSPI DSI Serialization Data Register (DSPI_SDR)* and the *Section 28.3.2.12: DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)* as the source of the serialized data. The DSPI_SDR holds the latest Parallel Input signal values which is sampled at every rising edge of the system clock. The DSPI_ASDR register is written by host software and used as an alternate source of serialized data.

DSPI_SSR register provides additional way to create the frame for transmission. Each bit from this register is OR'd with the TXSS bit and controls individual transmitted bit source. This way, the transmitted frame can have any combination of the DSPI_SDR and DSPI_ASDR bits. This feature allows control SPI based devices, requiring control and data fields in the frame. Control field may come from DSPI_ASDR register, set by the SoC CPU, while data field can be generated by SoC peripheral modules, like PWM timers.

A copy of the last 32-bit DSI frame shifted out of the Shift Register is stored in the *Section 28.3.2.13: DSPI DSI Transmit Comparison Register (DSPI_COMPR)*. This register

provides added visibility for debugging and it serves as a reference for transfer initiation control. *Figure 388* shows the DSI Serialization logic.

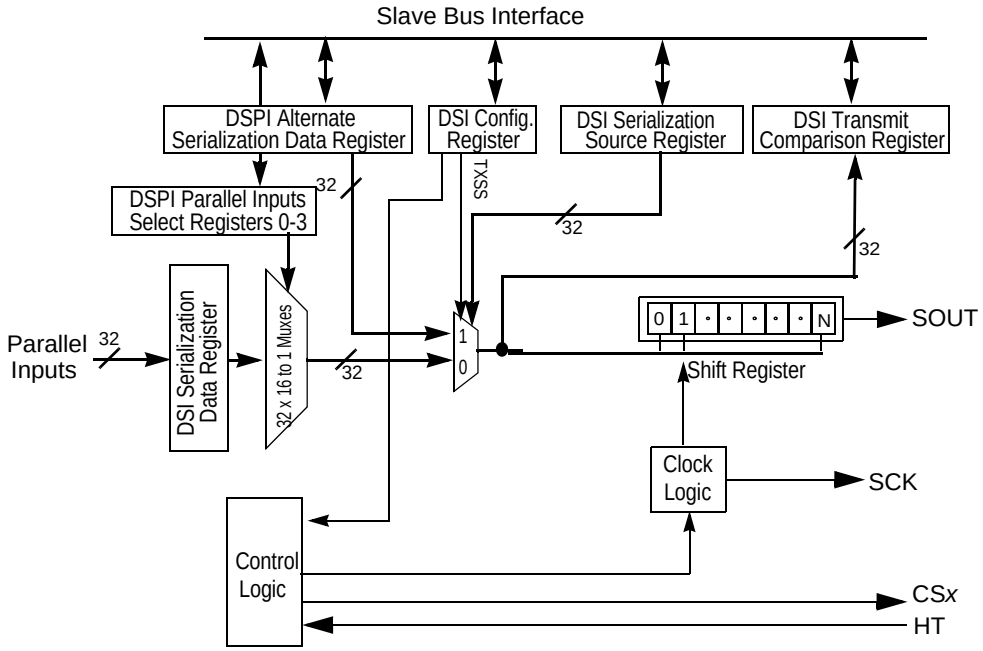


Figure 388. DSI Serialization Diagram

28.4.3.4 DSI Deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the *Section 28.3.2.14: DSPI DSI Deserialization Data Register (DSPI_DDR)*. The DSPI_DDR is memory mapped to allow host software to read the deserialized data directly.

The received data is bit-wise compared to the value of the DSI Deserialized Data Polarity Interrupt Register, bit-wise AND'ed with DSI Deserialized Interrupt Mask Register and the results OR'ed to produce DDIF flag in the DSPI_SR register. Which, in turn, can cause DDI interrupt request if the DSPI_RSER[DDIFRE] bit is set and/or stop DSI frame transmissions if the DMS bit of the DSPI_DSICR register is set.

Figure 389 shows the DSI Deserialization logic.

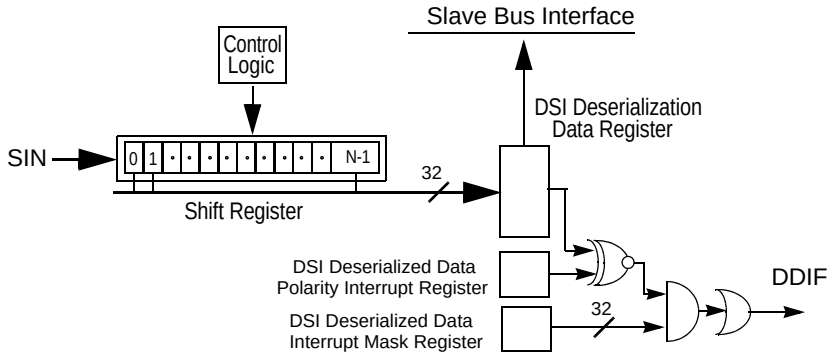


Figure 389. DSI Deserialization Diagram

28.4.3.5 DSI Transfer Initiation Control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. The transfer initiation conditions are selected by the TRRE and CID bits in the DSPI_DSICR. *Table 348* lists the four transfer initiation conditions.

Table 348. DSI Data Transfer Initiation Control

DSPI_DSICR Bits		Transfer Initiation Control
TRRE	CID	
0	0	Continuous
0	1	Change in Data
1	0	Triggered
1	1	Triggered or Change in Data

28.4.3.5.1 Continuous Control

For Continuous Control a new DSI frame shifts out when the previous transfer cycle has completed and the Delay after Transfer (t_{DT}) has elapsed.

28.4.3.5.2 Change In Data Control

For Change in Data Control a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI_COMPR. When the data, selected for the transfer from the DSPI_SDR and DSPI_ASDR registers is different from the data in the DSPI_COMPR a new DSI frame is transmitted. The MTRIG output signal is asserted every time a change in data is detected.

28.4.3.5.3 Triggered Control

For Triggered Control initiation of a transfer is controlled by the Hardware Trigger signal (HT). The TPOL bit in the DSPI_DSICR selects the active edge of HT. For HT to have any affect, the TRRE bit in the DSPI_DSICR must be set.

28.4.3.5.4 Triggered or Change In Data Control

For Triggered or Change in Data Control initiation of a transfer is controlled by the HT signal or by the detection of a change in data to be serialized.

28.4.3.6 Multiple Transfer Operation (MTO)

In DSI Configuration the MTO feature allows for multiple DSPIs within an SoC to be chained together in a parallel or serial configuration. The parallel chaining allows multiple DSPIs internal to an SoC and multiple SPI devices external to an SoC to share SCK and CS*x* signals thereby saving the SoC pins. The serial chaining allows bits from multiple DSPIs to be concatenated into a single DSI frame. MTO is enabled by setting the MTOE bit in the DSPI_DSICR.

In parallel and serial chaining there is one bus master and multiple bus slaves. The bus master initiates and controls the transfers, but the DSPI slaves generate trigger signals for the bus DSPI master when an internal condition in the slave warrants a transfer. The DSPI slaves also propagate triggers from other slaves to the master. When a DSPI slave detects a trigger signal on its HT input, the slave generates a trigger signal on the $\overline{\text{MTRIG}}$ output.

Serial and parallel chaining require multiplexing of signals external to the DSPI.

Note: TSB operation is not available in MTO mode. TSBC and MTOE bits of DSPI_DSICR register should not be set simultaneously.

28.4.3.6.1 Parallel Chaining

Parallel chaining allows multiple DSPIs internal to an SoC and multiple SPI/DSI devices external to an SoC to share common SCK and CS*x* signals thereby saving pins. Two pins are saved per pair of DSPI/SPI. *Figure 390* shows an example of how the blocks can be connected in an SoC.

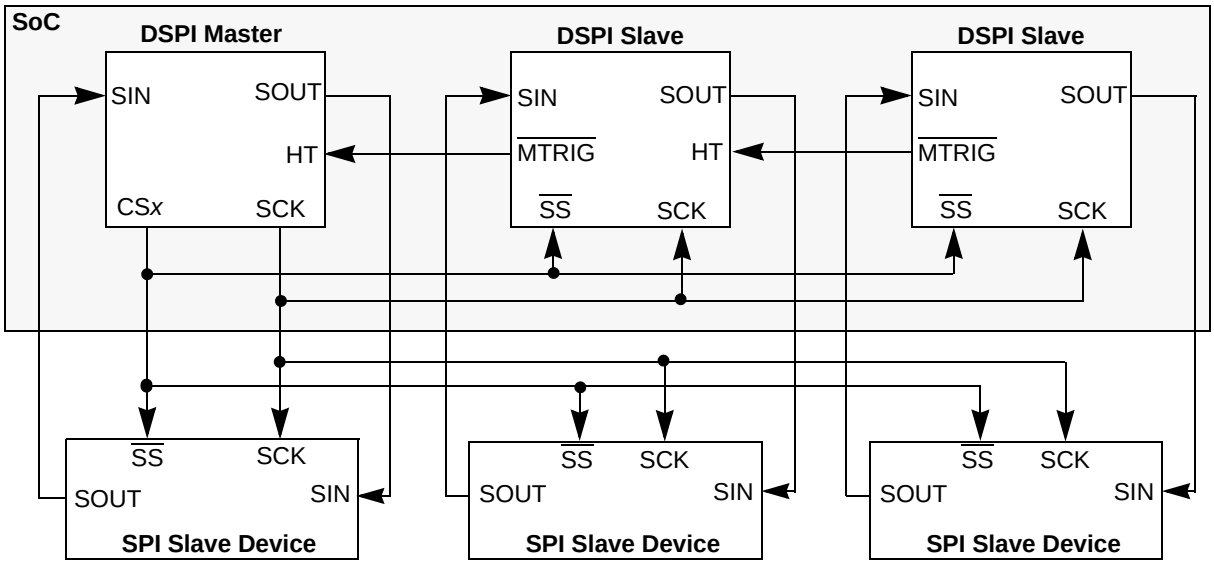


Figure 390. Example of Parallel Chaining of DSPIs

In parallel chaining, the SOUT and SIN of the three DSPIs connect to separate external SPI devices. All internal and external SPI blocks share CS*x* and SCK signals. DSPI0 controls and

initiates all transfers, but the DSPI slaves each have a trigger output signal MTRIG that indicates to DSPI0 that a trigger condition has occurred in the DSPI slaves. When the slave DSPI has a change in data to be serialized, it asserts the MTRIG signal that propagates to DSPI0 which initiates the transfer.

28.4.3.6.2 Serial Chaining

The serial chaining allows transfers of DSI frames of up to a total of 64 bits, using transfers of smaller DSI frames concatenated together by multiple DSPIs. *Figure 391* shows an example of how the blocks can be connected in an SoC.

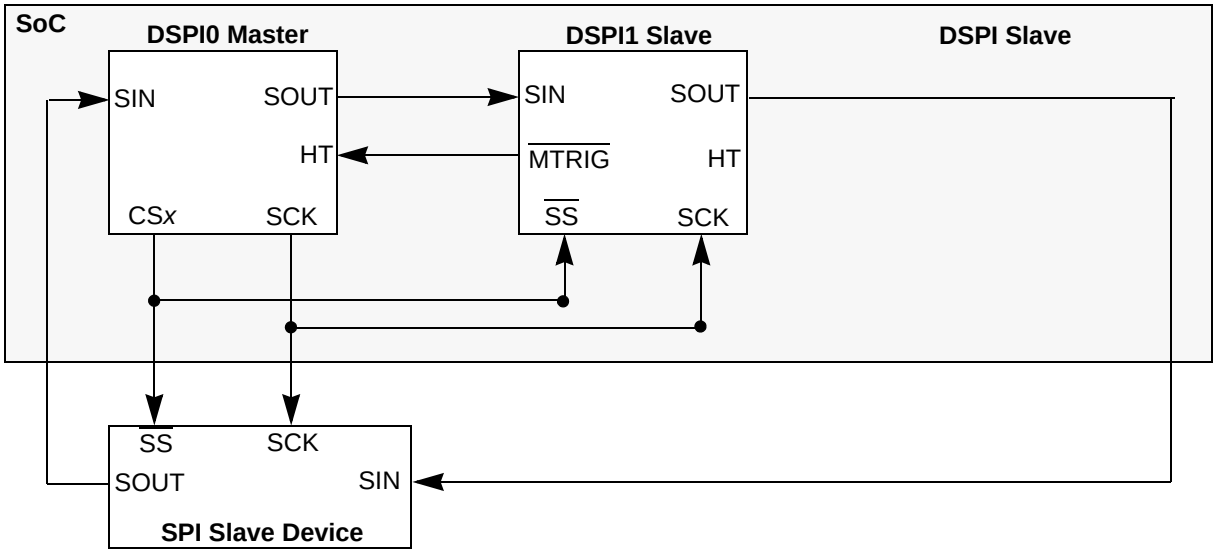


Figure 391. Example of Serial Chaining of DSPIs

In a serial chain, one DSPI block operates as a master, the other DSPI blocks operate as slaves. The data output (SOUT) of the master is connected to the data input (SIN) of the slave. The SOUT of a slave is connected to the SIN of subsequent slaves until the last block in the chain, where the SOUT is connected to an external pin, which connects to the input of an external SPI device. The slave DSPI and external SPI device use the master peripheral chip select (CS*x*) and clock (SCK).

The Trigger input of the master allows a slave DSPI to trigger a transfer when the data change occurs in the slave DSPI and the slave DSPI is operating in Change in Data mode. The Trigger input of the master is connected to MTRIG output of the slave.

The concatenated frames can be 8 to 64 bits long.

28.4.4 Combined Serial Interface (CSI) Configuration

The CSI Configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI Configuration allows interleaving of DSI data frames from the Parallel Input signals with SPI commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the Parallel Output signals or it is stored in the RX FIFO. The CSI Configuration allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI Configuration when the DCONF field in the DSPI_MCR is 0b10.

In CSI Configuration, the DSPI transfers DSI data based on *Section 28.4.3.4: DSI Deserialization*. When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so that the two CTAR registers associated with DSI data and SPI data assert different peripheral chip select signals denoted in the figure as PCSx and PCSy. The CSI Configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the Parallel Output signals. Data returned from the external slave while a SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

28.4.4.1 CSI Serialization

Serialization in the CSI configuration is similar to serialization in DSI Configuration. The transfer attributes for SPI frames are determined by the DSPI_CTAR register selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI_CTAR register selected by the DSICTAS field in the DSPI_DSICR.

The Parallel Inputs signal states are latched into the *Section 28.3.2.11: DSPI DSI Serialization Data Register (DSPI_SDR)* on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI_DSICR. When SPI frames are written to the TX FIFO they have priority over DSI data from the DSPI_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI_COMPR. The Transfer Priority Logic selects the source of the serialized data and asserts the appropriate CS*x* signal.

28.4.5 DSPI Baud Rate and Clock Delay Generation

The SCK 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 for doubling the baud rate. *Figure 392* shows conceptually how the SCK signal is generated.

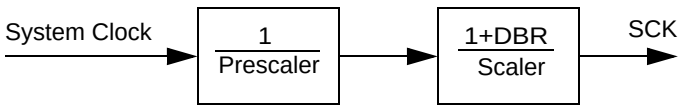


Figure 392. Communications Clock Prescalers and Scalers

28.4.5.1 Baud Rate Generator

The Baud Rate is the frequency of the Serial Communication Clock (SCK). The system clock is divided by a prescaler (PBR) and scaler (BR) to produce SCK with the possibility of halving the scaler division. The DBR, PBR and BR fields in the DSPI_CTAR registers select the frequency of SCK by the formula in the BR field description. *Table 349* shows an example of how to compute the baud rate.

Table 349. Baud Rate Computation Example

f_{sys}	PBR	Prescaler	BR	Scaler	DBR	Baud Rate
100 MHz	0b00	2	0b0000	2	0	25 Mb/s
20 MHz	0b00	2	0b0000	2	1	10 Mb/s

28.4.5.2 CS to SCK delay (t_{CSC})

The CS_*x* to SCK delay is the length of time from assertion of the CS_*x* signal to the first SCK edge. Refer to *Figure 394* for an illustration of the CS_*x* to SCK delay. The PCSSCK and CSSCK fields in the DSPI*x*_CTAR*n* registers select the CS_*x* to SCK delay, and the relationship is expressed by the following formula:

$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$

Table 350 shows an example of the computed CS to SCK delay.

Table 350. CS to SCK delay computation example

PCSSCK	Prescaler value	CSSCK	Scaler value	f_{SYS}	CS to SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

28.4.5.3 After SCK Delay (t_{ASC})

The After SCK Delay is the length of time between the last edge of SCK and the negation of PCS. See *Figure 394* and *Figure 395* for illustrations of the After SCK delay. The PASC and ASC fields in the DSPI_CTAR*x* registers select the After SCK Delay by the formula in the ASC field description. *Table 351* shows an example of how to compute the After SCK delay.

Table 351. After SCK Delay Computation Example

f_{sys}	PASC	Prescaler	ASC	Scaler	After SCK Delay
100 MHz	0b01	3	0b0100	32	0.96 µs

PCASC and ASC fields have no effect in TSB configuration.

28.4.5.4 Delay after Transfer (t_{DT})

The Delay after Transfer is the minimum time between negation of the signal for a frame and the assertion of the CS*x* signal for the next frame. See *Figure 394* for an illustration of the Delay after Transfer. The PDT and DT fields in the DSPI_CTAR*x* registers select the Delay after Transfer by the formula in the DT field description. *Table 352* shows an example of how to compute the Delay after Transfer.

Table 352. Delay after Transfer Computation Example

f_{sys}	PDT	Prescaler	DT	Scaler	Delay after Transfer
100 MHz	0b01	3	0b1110	32768	0.98 ms

When in non-continuous clock mode the t_{DT} delay is configured according *Equation 12*. When in continuous clock mode and TSB is not enabled the delay is fixed at 1 SCK period.

In TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See detailed information on *Section 28.4.9: Timed Serial Bus (TSB)*.

28.4.5.5 Peripheral Chip Select Strobe Enable (CS5_*x*)

The CS5_*x* signal provides a delay to allow the CS*x* signals to settle after a transition occurs thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI_MCR, CS5_*x* provides a signal for an external demultiplexer to decode the CS4_*x* and signals into as many as 32 glitch-free CS*x* signals. *Figure 393* shows the timing of the CS5_*x* signal relative to CS*x* signals.

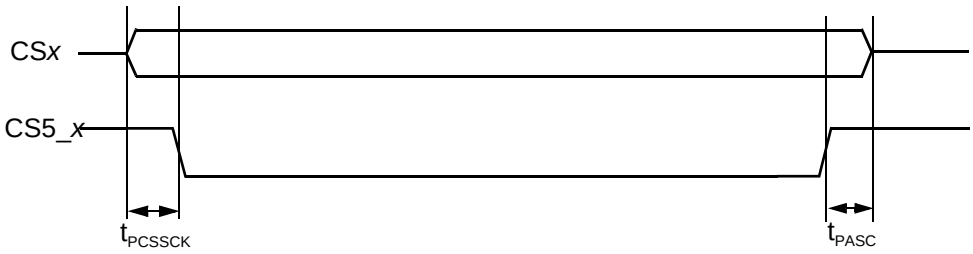


Figure 393. Peripheral Chip Select Strobe Timing

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_CTAR based on the following formula:

Equation 14

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between CS5_*x* negation and PCS negation is selected by the PASC field in the DSPI_CTAR based on the following formula:

Equation 15

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 353 shows an example of how to compute the t_{pcssck} delay.

Table 353. Peripheral Chip Select Strobe Assert Computation Example

f_{sys}	PCSSCK	Prescaler	Delay before Transfer
100 MHz	0b11	7	70.0 ns

Table 354 shows an example of how to compute the t_{pasc} delay.

Table 354. Peripheral Chip Select Strobe Negate Computation Example

f_{sys}	PASC	Prescaler	Delay after Transfer
100 MHz	0b11	7	70.0 ns

The CS5_*x* signal is not supported when Continuous Serial Communication SCK or TSB mode are enabled.

28.4.6 Transfer Formats

The SPI serial communication is controlled by the Serial Communications Clock (SCK) signal and the CS*x* signals. The SCK signal provided by the master device synchronizes shifting and sampling of the data on the SIN and SOUT 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_CTARx) select the polarity and phase of the serial clock, SCK. The polarity bit selects the idle state of the SCK. The clock phase bit selects if the data on SOUT is valid before or on the first SCK edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI_CTAR0 (SPI) or DSPI_CTAR1 (DSI) 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 and the slave devices 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_MCR selects between Classic SPI Format and Modified Transfer Format.

In the SPI and DSI Configurations, the DSPI provides the option of keeping the CS*x* signals asserted between frames. See *Section 28.4.6.5: Continuous Selection Format*, for details.

28.4.6.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in *Figure 394* 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 pins on the odd-numbered SCK edges and change the data on their SOUT pins on the even-numbered SCK edges.

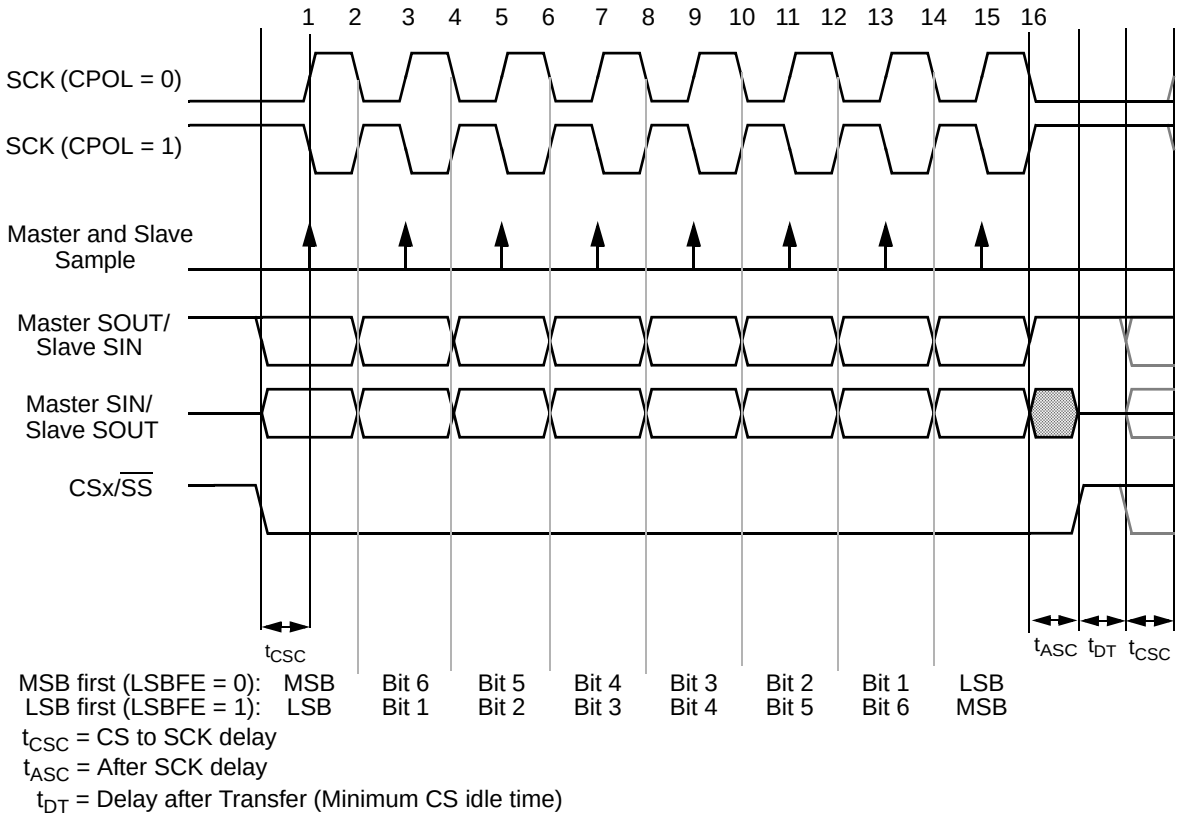
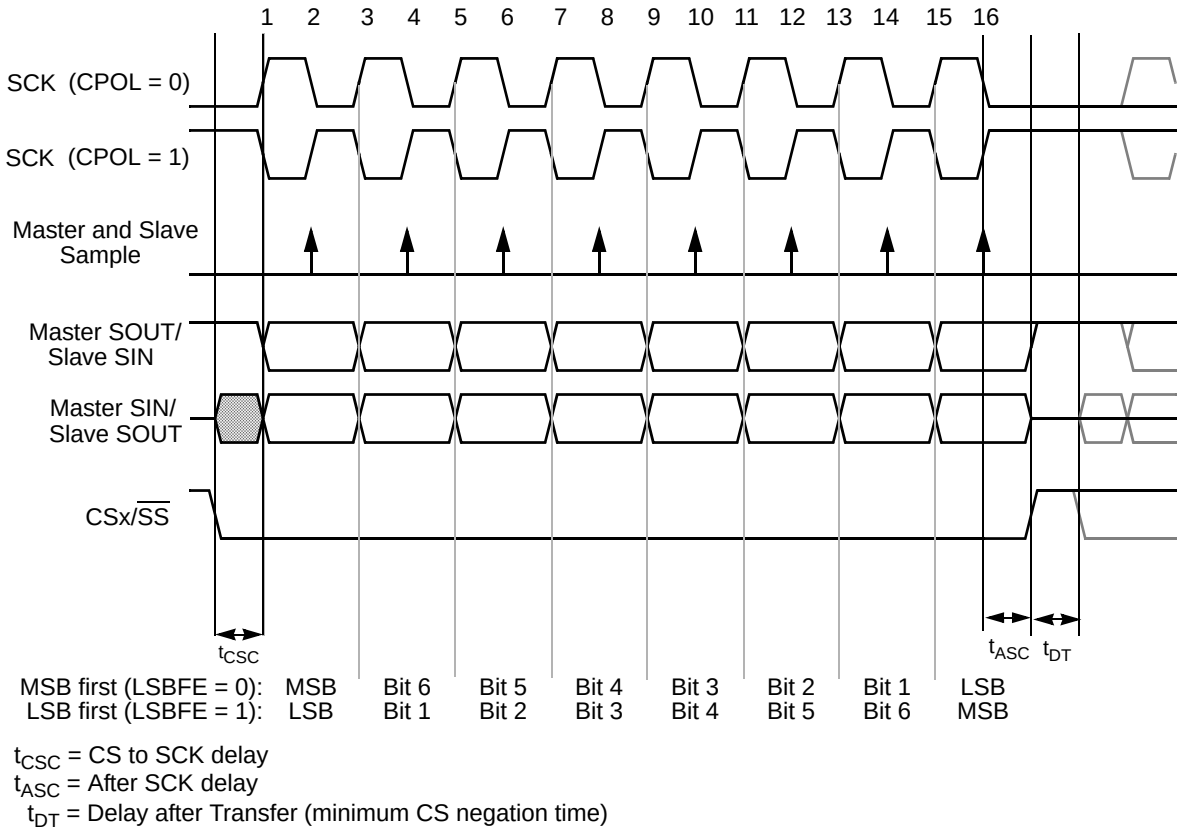


Figure 394. DSPI Transfer Timing Diagram (MTFE=0, CPHA=0, FMSZ=8)

The master initiates the transfer by placing its first data bit on the SOUT 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 pin. After the t_{CSC} delay elapses, the master outputs the first edge of SCK. The master and slave devices use this edge to sample the first input data bit on their serial data input signals. At the second edge of the SCK 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 pins on the odd-numbered clock edges and changes the data on their SOUT 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* signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

28.4.6.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in *Figure 395* is used to communicate with peripheral SPI slave devices that require the first SCK edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT pins on the odd-numbered SCK edges and sample the data on their SIN pins on the even-numbered SCK edges



t_{CSC} = CS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 395. 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 edge and at the same time places valid data on the master SOUT pin. The slave responds to the first SCK edge by placing its first data bit on its slave SOUT pin.

At the second edge of the SCK the master and slave sample their SIN pins. For the rest of the frame the master and the slave change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN 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.

28.4.6.3 Modified SPI/DSI 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 tolerate more 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.

The master and the slave place data on the SOUT pins at the assertion of the CS*x* signal. After the CS to SCK delay has elapsed the first SCK edge is generated. The slave samples the master SOUT signal on every odd numbered SCK edge. The DSPI in the slave mode when the MTFE bit is set also places new data on the slave SOUT on every odd numbered clock edge. Regular external slave, configured with CPHA=0 format drives its SOUT output at every even numbered SCK clock edge.

The DSPI master places its second data bit on the SOUT line one system clock after odd numbered SCK edge if the system frequency to SCK frequency ratio is higher than three. If this ratio is below four the master changes SOUT at odd numbered SCK edge. The point where the master samples the SIN is selected by the DSPI_MCR[SMPL_PT] field. The *Table 325* lists the number of system clock cycles between the active edge of SCK and the master Sample point. The master sample point can be delayed by one or two system clock cycles. The SMPL_PT field should be set to 0 if the system to SCK frequency ratio is less than 4.

Following timing diagrams illustrate the DSPI operation with MTFE=1. Timing delays shown are:

- T_{csc} - CS to SCK assertion delay
- T_{acs} - After SCK delay
- T_{su_ms} - master SIN setup time
- T_{hd_ms} - master SIN hold time
- T_{vd_sl} - slave data output valid time, time between slave data output SCK driving edge and data becomes valid.
- T_{su_sl} - data setup time on slave data input
- T_{hd_sl} - data hold time on slave data input
- T_{sys} - system clock period.

Figure 396 shows the modified transfer format for CPHA = 0 and Fsys/Fsck = 4. Only the condition where CPOL = 0 is illustrated. Solid triangles show the data sampling clock edges. The two possible slave behavior are shown.

- Signal, marked “SOUT of Ext Slave”, presents regular SPI slave serial output.
- Signal, marked “SOUT of DSPI Slave”, presents DSPI in the slave mode with MTFE bit set.

Other MTFE = 1 diagrams show DSPI SIN input as being driven by a regular external SPI slave, configured according DSPI master CPHA programming.

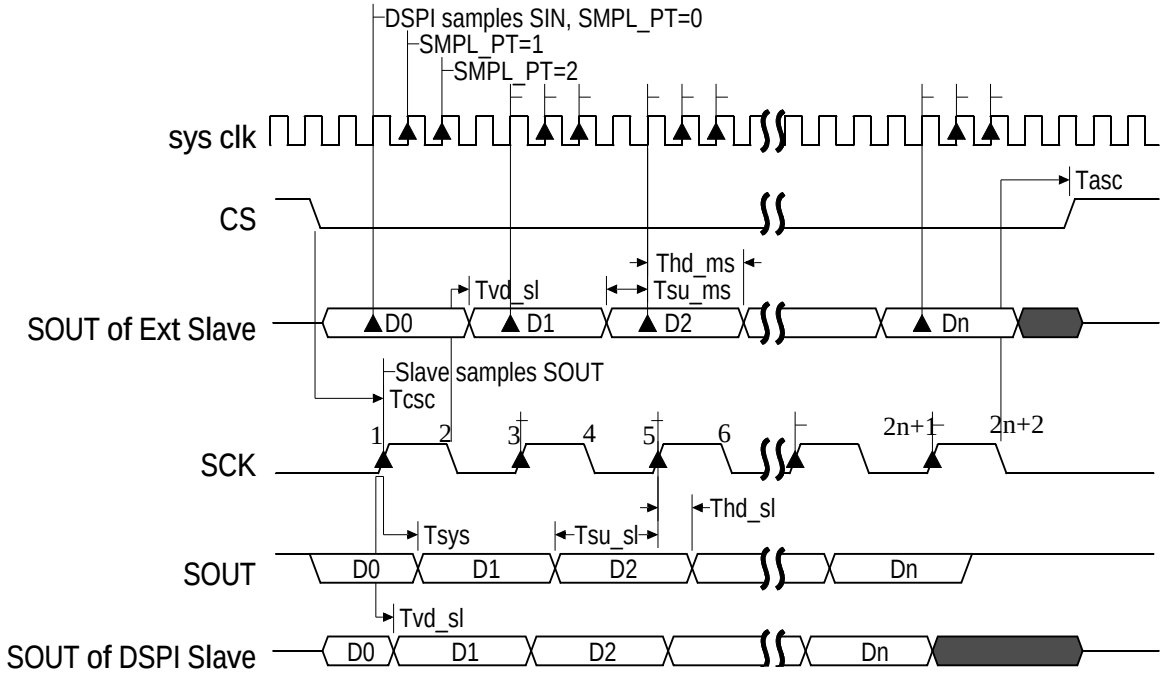


Figure 396. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/4$)

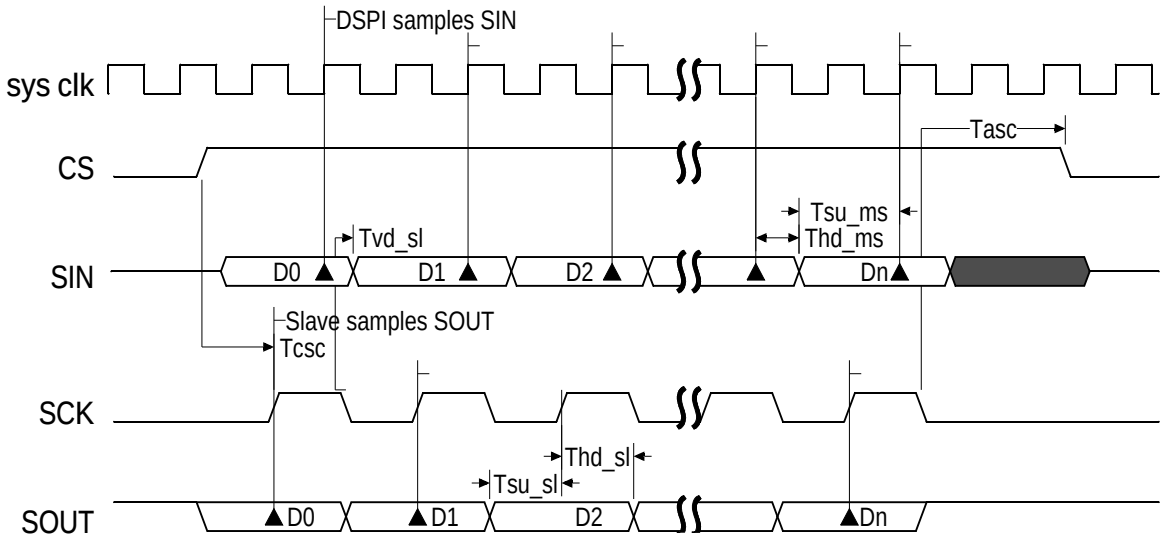


Figure 397. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/2$)

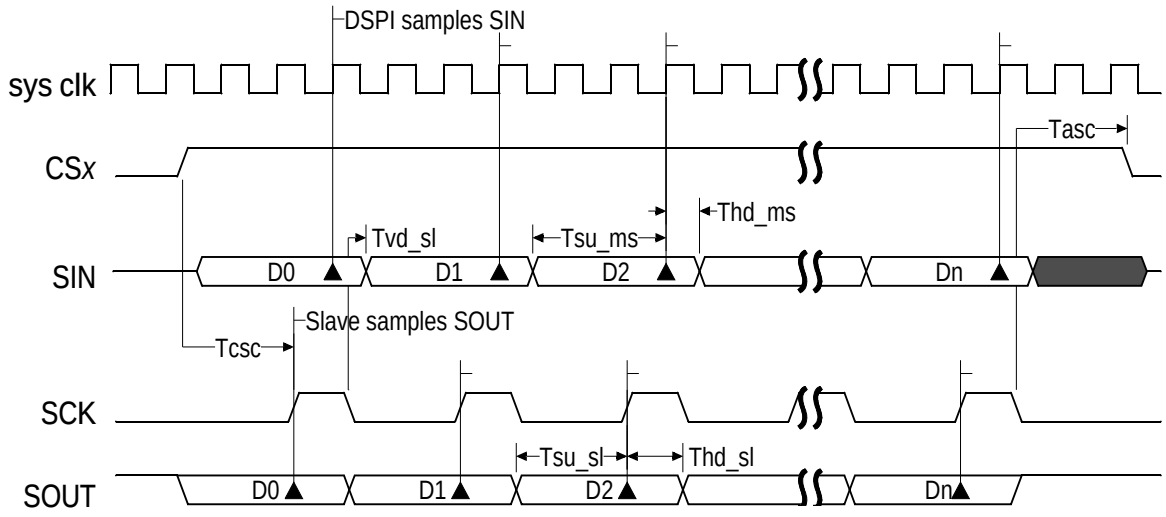


Figure 398. DSPI Modified Transfer Format (MTFE=1, CPHA=0, $f_{sck} = f_{sys}/3$)

28.4.6.4 Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1)

Figure 399 shows the Modified Transfer Format for CPHA = 1. Only the condition, where CPOL = 0 is shown. At the start of a transfer the DSPI asserts the CS*x* 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 will be 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.

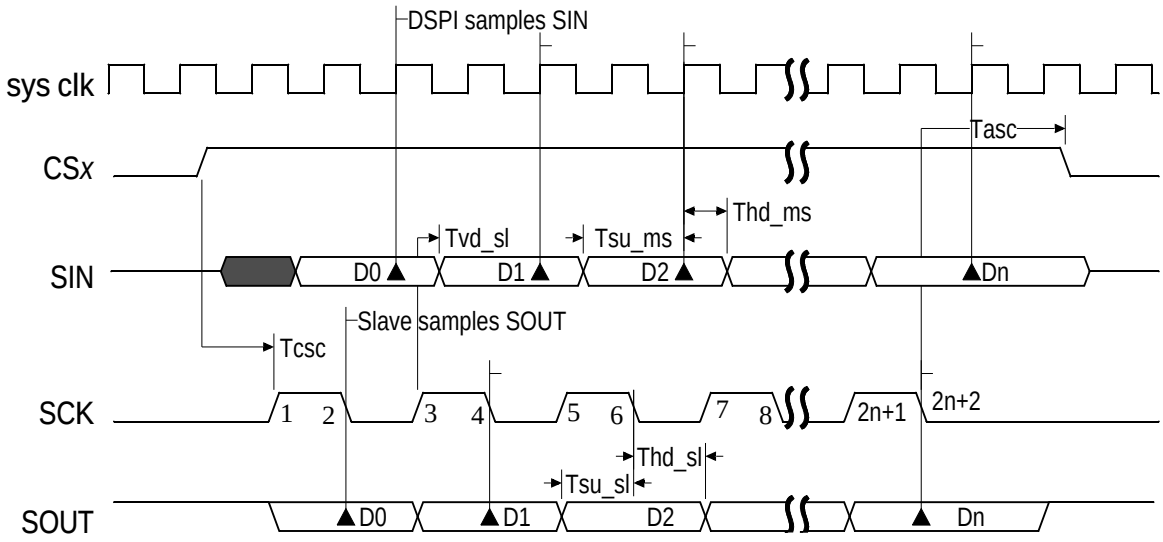


Figure 399. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/2$)

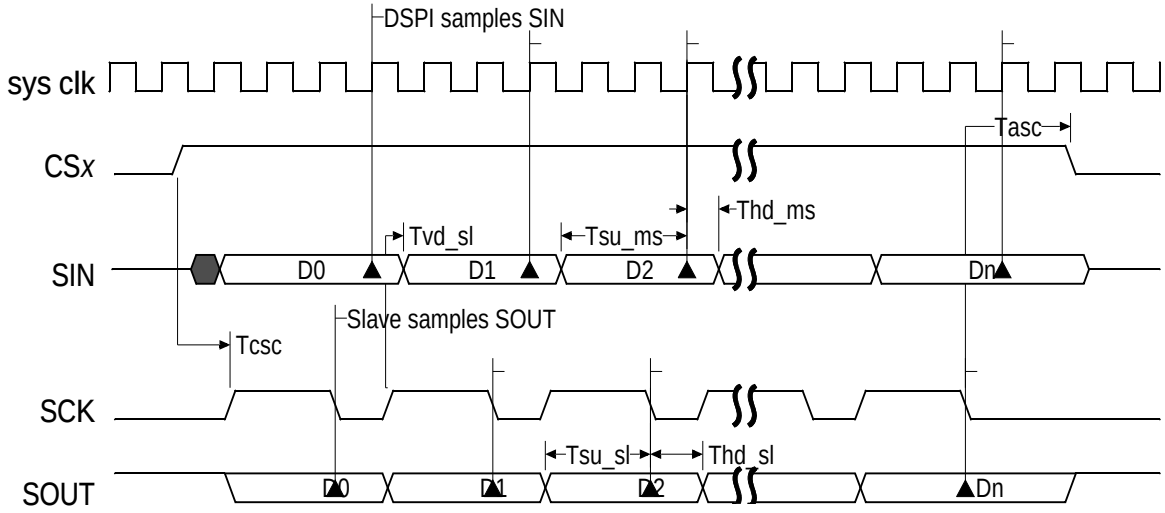


Figure 400. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/3$)

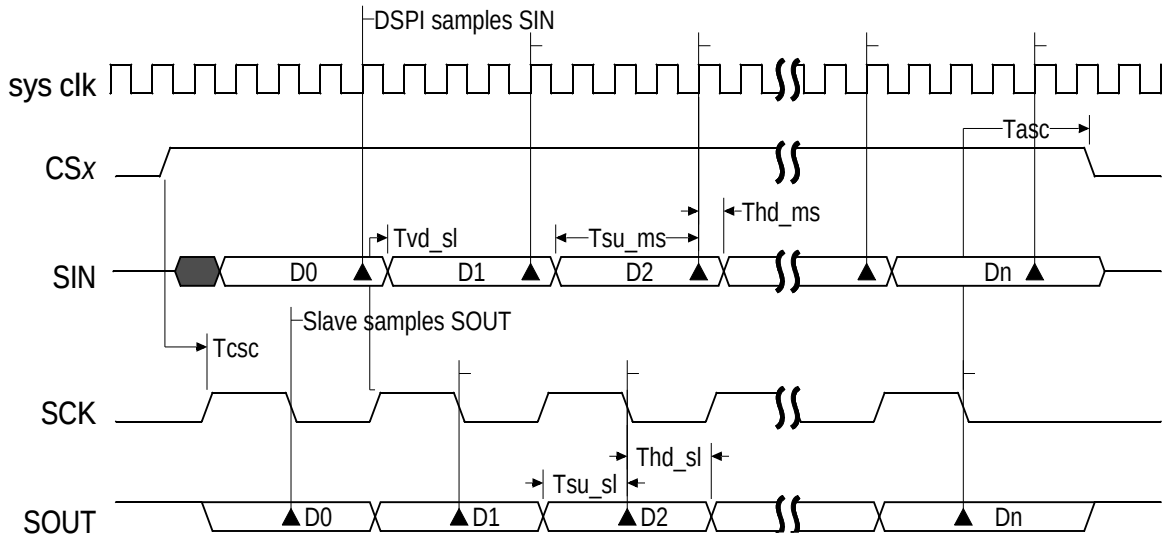
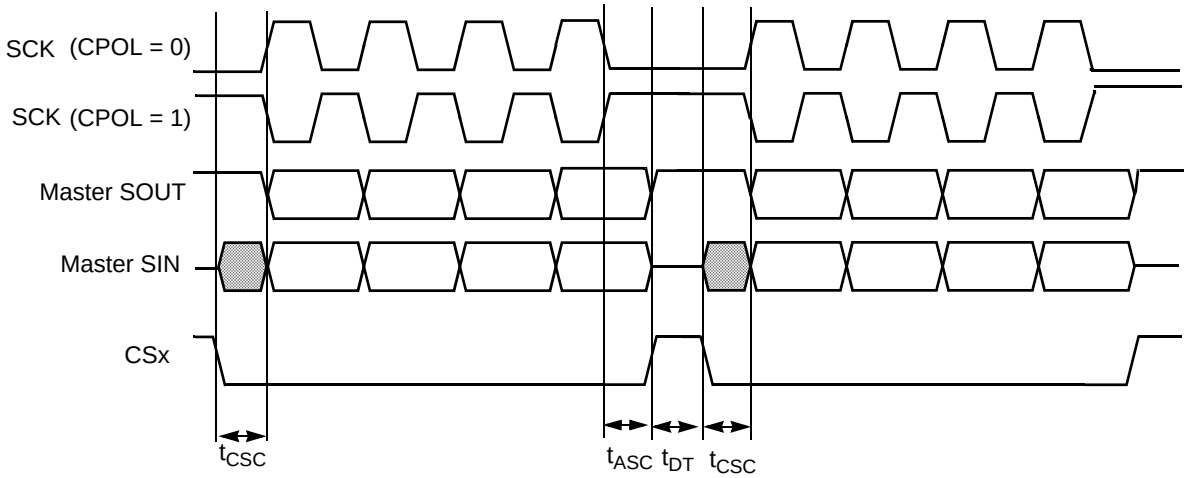


Figure 401. DSPI Modified Transfer Format (MTFE=1, CPHA=1, $f_{sck} = f_{sys}/4$)

28.4.6.5 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. Continuous Selection is enabled for the DSI Configuration by setting the DCONT bit in the DSPI_DSICR. The behavior of the CS*x* signals in the two configurations is identical so only SPI Configuration will be described.

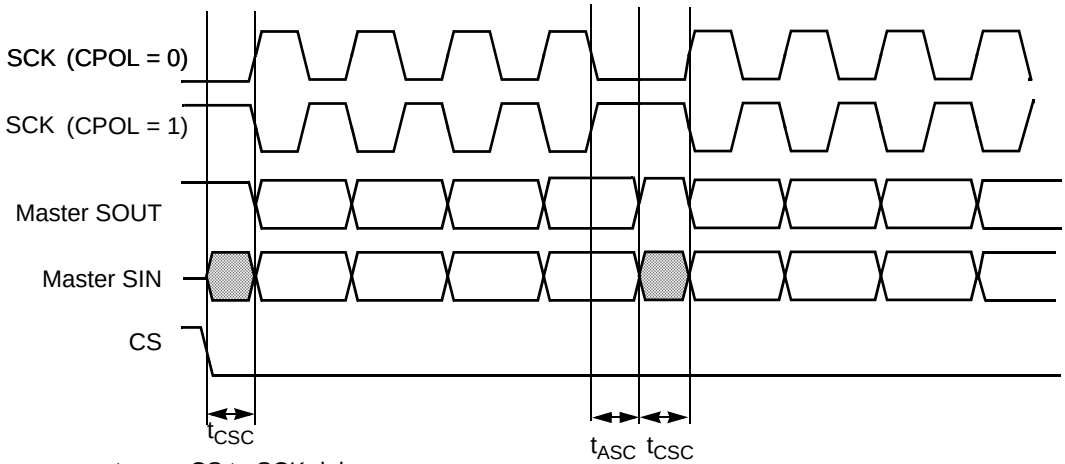
When the CONT bit = 0, the DSPI drives the asserted Chip Select signals to their idle states in between frames. The idle states of the Chip Select signals are selected by the PCSISn bits in the DSPI_MCR. *Figure 402* shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.



t_{CSC} = CS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 402. Example of Non-Continuous Format (CPHA=1, CONT=0)

When the CONT bit = 1, the PCS signal remains asserted for the duration of the two transfers. The Delay between Transfers (t_{DT}) is not inserted between the transfers. *Figure 403* shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 1.



t_{CSC} = CS to SCK delay
t_{ASC} = After SCK delay

Figure 403. Example of Continuous Transfer (CPHA=1, CONT=1)

When using DSPI with continuous selection follow these rules:

- all transmit commands must have the same PCSn bits programming
- the DSPI_CTARs, selected by transmit commands, must be programmed with the same transfer attributes. Only FMSZ field can be programmed differently in these DSPI_CTARs.

Note: User 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.

Note: 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.

28.4.7 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_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA=1. Clearing CPHA 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.
- When the DSPI is in DSI configuration, always use the CTAR specified by the DSICTAS field.
- 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.

It is recommended to keep the baud rate the same 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 the External Stop mode or Module Disable mode.

Enabling Continuous SCK disables the CS to SCK delay and the Delay after Transfer (t_{DT}) is fixed to one $_{SCK}$ cycle. When TSB configuration is enabled the t_{DT} is programmable from 1 to 65 $_{SCK}$ cycles. *Figure 404* shows timing diagram for Continuous SCK format with Continuous Selection disabled.

Note: When in Continuous SCK mode, for the SPI transfer CTAR0 should always be used, and the TX-FIFO must be clear using the MCR.CLR_TXF field before initiating transfer.

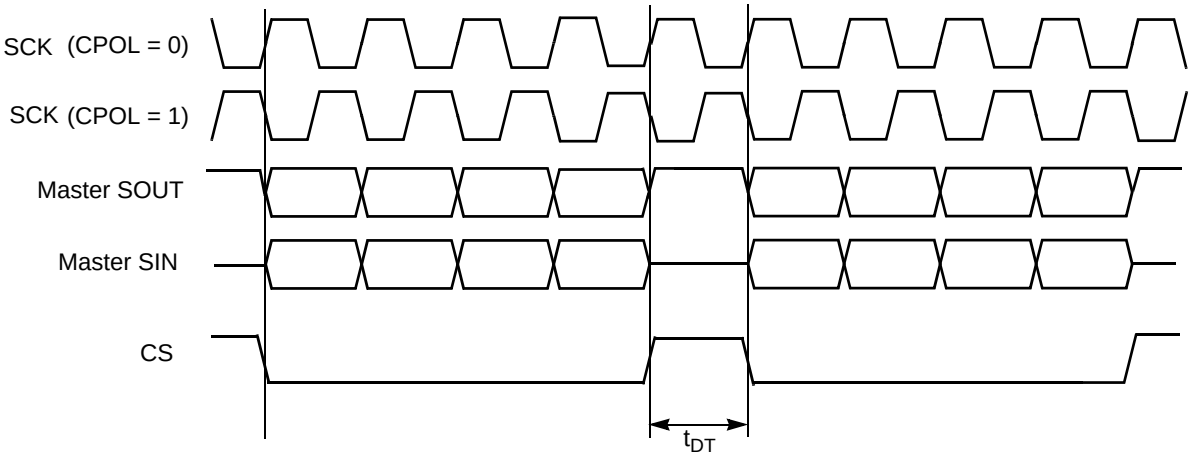


Figure 404. Continuous SCK Timing Diagram (CONT=0)

If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPI_DSICR is set, CS*x* remains asserted between the transfers. Under certain conditions, SCK can continue with CS*x* asserted, but with no data being shifted out of SOUT (SOUT pulled high). This can cause the slave to receive incorrect data. Those conditions include:

- Continuous SCK with CONT bit set, but no data in the transmit FIFO.
- Continuous SCK with CONT bit set and entering STOPPED state (refer to *Section 28.4.1: Start and Stop of DSPI Transfers*).
- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode.

Figure 405 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

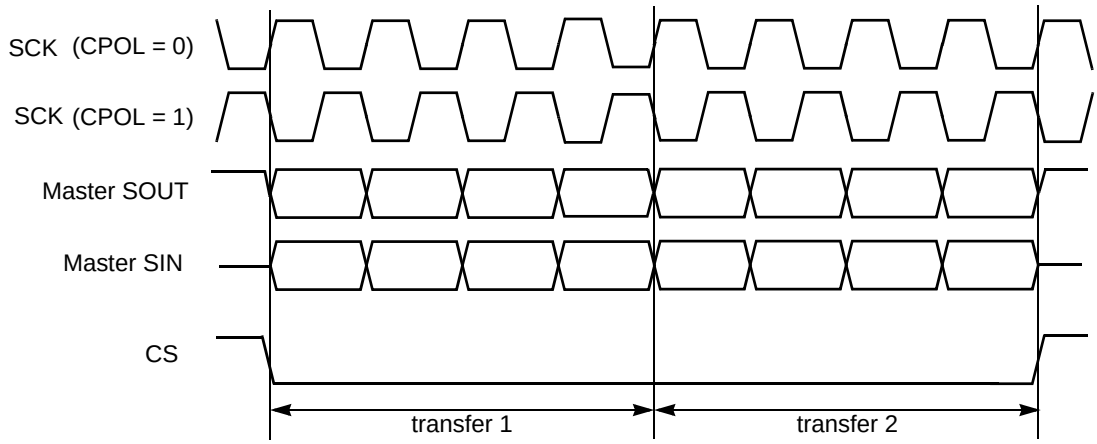


Figure 405. Continuous SCK Timing Diagram (CONT=1)

28.4.8 Slave Mode Operation Constraints

Slave mode logic shift register is buffered. This allows data streaming operation, when the DSPI is permanently selected and data is shifted in with a constant rate.

The transmit data is transferred at second SCK clock edge of the each frame to the shift register if the $\overline{SS}$ signal is asserted and any time when transmit data is ready and $\overline{SS}$ signal is negated.

Received data is transferred to the receive buffer at last SCK edge of each frame, defined by frame size programmed to the CTAR0/1 register. Then the data from the buffer is transferred to the RXFIFO or DDR register.

If the $\overline{SS}$ negates before that last SCK edge, the data from shift register is lost.

This buffering scheme allows to operate slave clock with higher frequency than the system frequency. The clocks relationship is defined by *Equation 16*. *FrameSize* is the value of the CTAR0/1[FMSZ] field plus one.

Equation 16 $f_{SCK} < f_{SYS} \times FrameSize/3$

In slave mode, the DSPI performance is limited by synchronization and propagation through combinational logic to the serial data output. Performance limitations for various slave configurations are summarized in *Table 355*.

CPHA	System Constraint(1)
0	$t_{CSC} > 2 \times T_{SYS} + t_{sl_vd} + t_{ms_su}$
1	$t_{CSC} > 2 \times T_{SYS}$

Table 355. DSPI Slave Mode System Constraints

1. Timing parameter definitions are:
   T_{SYS} = system clock period
   t_{CSC} = delay from PCS asserted to first serial clock edge
   t_{sl_vd}= SOUT data valid time in slave mode
   t_{ms_su} = SPI master data setup time

These limitations must be taken into account at a system level with the SPI transfer timing set to allow for worst case performance of the SPI slave (including I/O and board level propagation delays). To ensure correct operation, the master SPI device must be configured so that tCSC are greater the requirements shown in the *Table 355* and the SCK clock frequency should comply to the *Equation 16*.

28.4.9 Timed Serial Bus (TSB)

The DSPI can be programmed in Timed Serial Bus configuration by setting the TSBC bit in the DSPI_DSICR register. See *Section 28.3.2.10: DSPI DSI Configuration Register (DSPI_DSICR)* for details.

TSB configuration provides the Micro Second Channel (MSC) downstream channel support.

The MSC upstream channel is not supported by the DSPI, but can be supported by any available Serial Communication Controller (SCI or UART) in the SoC.

To work in TSB mode the DSPI must be in master mode and in DSI (DCONF = 0b01) or CSI (DCONF = 0b10) configuration. Both Continuous and Non Continuous Serial Communication Clock (controlled by the DSPI_MCR[CONT_SCKE] bit) are supported in the TSB mode.

Figure 406 shows the signals used in the TSB interface.

In the TSB configuration the DSPI is able to send from 4 to 34 bits MSC data frames (4 to 32 serialized data bits and up to 2 Data Selection zero bits). The serialized data bits source can be either:

- the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), written by the host software,
- Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR).

DSPI_DSICR TXSS bit or DSPI_SSR register bits define the source of the data.

The Least Significant Bits of the DSPI_ASDR or DSPI_SDR registers are selected to be serialized if the data frame is set to less than 32 bits.

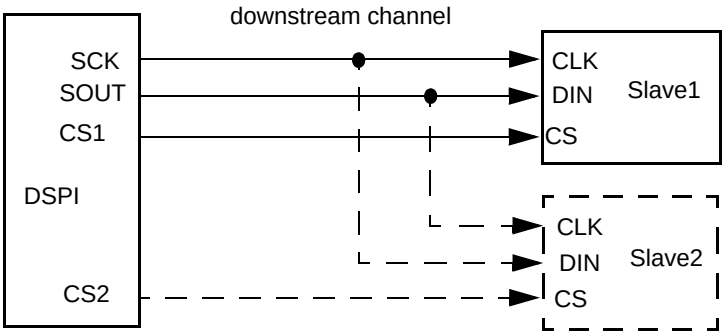


Figure 406. DSPI usage in the TSB Configuration

The CS*x* signals are driven together with SOUT. The t_{CSC} and t_{ASC} delays are not available. Delay after Transfer (DT) is set in SCK clock periods as a binary number formed by concatenation of the DSPI_CTARn PDT and DT fields plus one, allowing to set DT from 1 to 64 serial clock periods. DT field provides least significant bits and PDT field provides most significant bits of the Delay after Transfer.

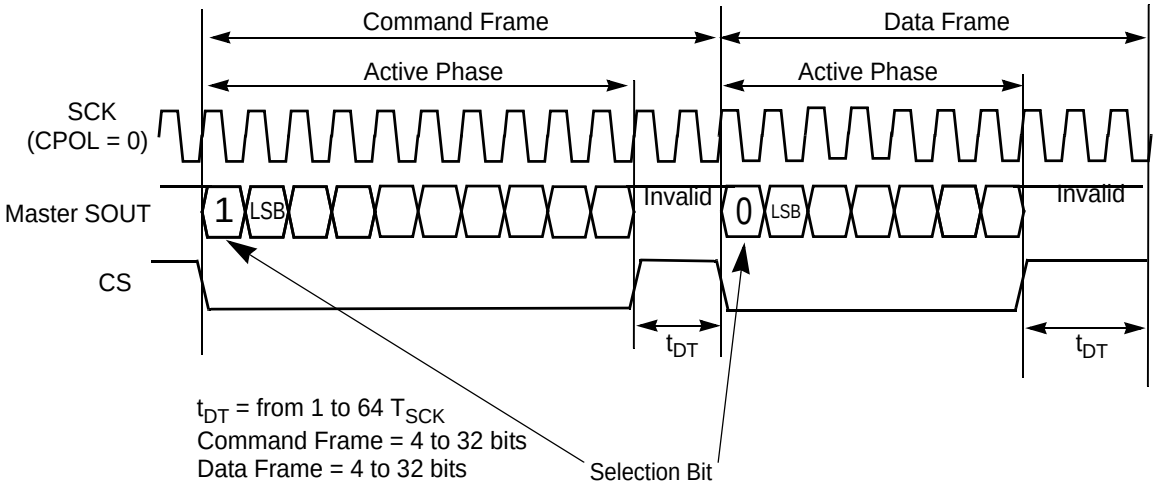


Figure 407. TSB Downstream Frames

The *Figure 407* shows the two types of MSC downstream frames - command frame, and data frame.

The first transmitted bit, called the selection bit, determines the frame type:

- The selection bit “0” indicates a data frame
- The selection bit “1” indicates a command frame

Data frame may contain up to 2 selection bits to support two external slave devices, (so called dual receiver configuration) or no selection bits at all.

The command frame can be written by software, through SPI TX FIFO, using one or two FIFO entries with help of the CONT bit. The data frame consists of up to 32 bits from the SDR or ASDR registers and up to two zero selection bits. Number of data bits in the data frame is defined by the DSCICR1[TSBCNT] field.

The selection bit of the MSC command frames (1) can be implemented by software.

The selection bits in the data frames are enabled by DSPI_DSICR1 DSE0 and DSE1 bits. Each DSEn bit set increases the data frame size by one bit.

To comply with MSC specification, set DSPI_CTARn[LSBFE] to transmit least significant bit first.

Regardless the LSBFE bit setting, the Data Frame Selection Bits, if enabled, are always transmitted first, before corresponded data sub frames.

28.4.9.1 MSC Dual Receiver Support with PCS Switch Over

When in TSB mode it is possible to switch the set of CS*x* signals that are driven during the first part of the frame to a different set of CS*x* signals during the second part of the frame. The bit, at which this switch over occurs, is defined by FMSZ field of the DSPI_CTARn register, selected by DSICTAS field of the DSICR register.

Number of the bits, not including the Data Selection Bit, in the first part of the frame is equal to value of the FMSZ field plus one. During this part of the frame the CS*x* signal levels are controlled by DSPI_DSICR DPCSn bits, after that by DSPI_DSICR1 DPCS1_n bits.

The PCS switch over occurs at driving edge of the SCK clock output.

The second Data Selection Bit is inserted after the PCS switch over if enabled.

Data Frame with PCS switch over is shown in *Figure 408*.

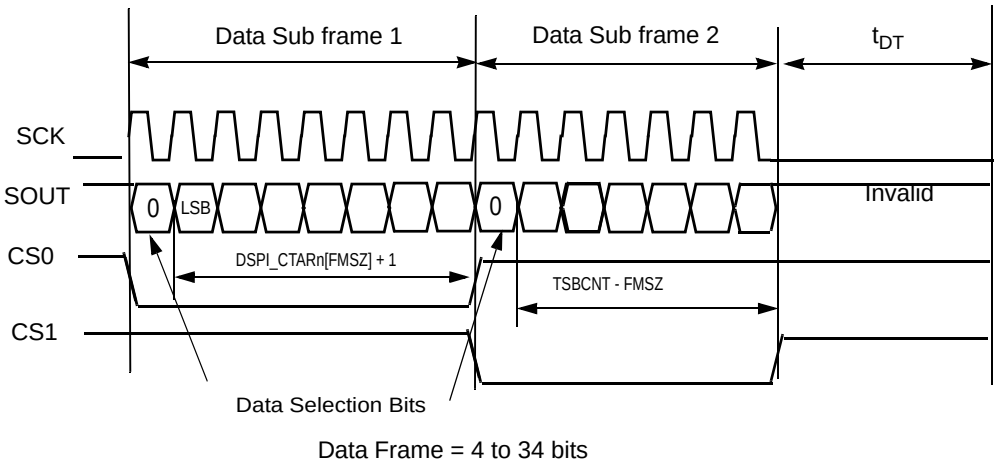


Figure 408. TSB Data Frame Format for MSC Dual Receiver Operation

28.4.10 Parity Generation and Check.

The DSPI module can generate and check parity in the serial frame. The parity bit replaces the last transmitted bit in the frame. The parity is calculated for all transmitted data bits in frame, not including the last, would be transmitted, data bit. The parity generation/control is done on frame basis. The registers fields, setting frame size defines the total number of bits in the frame, including the parity bit. Thus, to transmit/receive the same number of data bits with parity check, increase the frame size by one versus the same data size frame without the parity check.

Parity can be selected as odd or even. Parity Errors in the received frame set Parity Error flags in the Status register. The Parity Error Interrupt Requests are generated if enabled. The DSPI module can be programmed to stop SPI or/and DSI frame transmission in case of a frame reception with parity error.

28.4.10.1 Parity for SPI Frames

When the DSPI is in the master mode the parity generation is controlled by PE and PP bits of the TX FIFO entries (DSPI_PUSHR). Setting the PE bit enables parity generation for transmitted SPI frames and parity check for received frames. PP bit defines polarity of the parity bit.

When continuous PCS selection is used to transmit SPI data, two parity generation scenarios are available:

- Generate/check parity for the whole frame
- Generate/check parity for each sub-frame separately.

To generate/check parity for the whole frame set PE bit only in the last command/TX FIFO entry, forming this frame (with the DSPI_PUSHR register).

To generate/check parity for each sub-frame set PE bit in each command/TX FIFO entry, forming this frame.

If the parity error occurs for received SPI frame, the DSPI_SR[SPEF] bit is set. If DSPI_MCR[PES] bit is set, the DSPI stops SPI frames transmission. To resume SPI operation clear the DSPI_SR[SPEF] or the DSPI_MCR[PES] bits.

In slave mode the parity is controlled by the PE and PP bits of the DSPI_CTAR0 register similar to the master mode parity generation without continuous PCS selection.

28.4.10.2 Parity for DSI Frames

Parity generation is controlled by PE and PP bits of the DSPI_DSICR register similar to the SPI frames. The parity is calculated and checked for each DSI frame. (DSPI_DSICR[DCONT] bit has no effect on parity generation.)

If the parity error occurs for received DSI frame, the DSPI_SR[DPEF] bit is set. If DSPI_DSICR[PES] bit is set, the DSPI stops DSI frames transmission. To resume DSI operation clear the DSPI_SR[DPEF] or the DSPI_DSICR[PES] bits.

28.4.11 Interrupts/DMA Requests

The DSPI has several conditions that can only generate interrupt requests and two conditions that can generate interrupt or DMA request. *Table 356* lists these conditions.

Table 356. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of Queue (EOQ)	EOQF	X	
TX FIFO Fill	TFFF	X	X
Transfer Complete	TCF	X	
TX FIFO Underflow	TFUF	X	
RX FIFO Drain	RFDF	X	X
RX FIFO Overflow	RFOF	X	
SPI Parity Error	SPEF	X	
DSI Parity Error	DPEF	X	
DSI Data Interrupt	DDIF	X	

Each condition has a flag bit in the *Section 28.3.2.4: DSPI Status Register (DSPI_SR)* and an Request Enable bit in the *Section 28.3.2.5: DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)*. The TX FIFO Fill Flag (TFFF) and RX FIFO Drain Flag (RFDF) generate interrupt requests or DMA requests depending on the TFFFDIRS and RFDFDIRS bits in the DSPI_RSER.

The DSPI module also provides a global interrupt request line, which is asserted when any of individual interrupt requests lines is asserted.

28.4.11.1 End of Queue Interrupt Request

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 set and the DSPI_RSER[EOQFRE] bit is set.

28.4.11.2 Transmit FIFO Fill Interrupt or DMA Request

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 DSPI_RSER[TFFFRE] bit is set. The

DSPI_RSER[TFFFDIRS] bit selects whether a DMA request or an interrupt request is generated.

28.4.11.3 Transfer Complete Interrupt Request

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 DSPI_RSER[TCFRE] bit is set.

28.4.11.4 Transmit FIFO Underflow Interrupt Request

The Transmit FIFO Underflow Request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for the DSPI, operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the DSPI_RSER[TFUFRE] bit is set, an interrupt request is generated.

28.4.11.5 Receive FIFO Drain Interrupt or DMA Request

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 DSPI_RSER[RFDFRE] bit is set. The DSPI_RSER[RFDFDIRS] selects whether a DMA request or an interrupt request is generated.

28.4.11.6 Receive FIFO Overflow Interrupt Request

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 DSPI_RSER[RFOFRE] bit must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI_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 cleared, the incoming data is ignored.

28.4.11.7 SPI Frame Parity Error Interrupt Request

The SPI Frame Parity Error Flag indicates that a SPI frame with parity error had been received. The DSPI_RSER[SPEFRE] bit must be set for the interrupt request to be generated.

28.4.11.8 DSI Frame Parity Error Interrupt Request

The DSI Frame Parity Error Flag indicates that a DSI frame with parity error has been received. The DSPI_RSER[DPEFRE] bit must be set for the interrupt request to be generated.

28.4.12 Power Saving Features

The DSPI supports two power-saving strategies:

- External Stop mode
- Module Disable mode - Clock gating of non-memory mapped logic

28.4.12.1 Stop Mode (External Stop Mode)

The DSPI supports the stop mode protocol. When a request is made to enter external stop mode, the DSPI block acknowledges the request. If a serial transfer is in progress, the DSPI waits until it reaches the frame boundary before it is ready to have its clocks shut off.While the clocks are shut off, the DSPI memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in External Stop mode.

28.4.12.2 Module Disable Mode

Module disable mode is a block-specific mode that the DSPI can enter to save power. Host CPU can initiate the module disable mode by setting the MDIS bit in the DSPI_MCR.

When the MDIS bit is set, the DSPI negates Clock Enable signal at the next frame boundary. If implemented, the Clock Enable signal can stop the clock to the non-memory mapped logic. When Clock Enable is negated, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different effect 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 has no effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI_MCR have no effect 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 effect. Writing to the DSPI_TCR during module disable mode has no effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

28.5 Initialization/Application Information

28.5.1 How to Manage DSPI Queues

The queues are not part of the DSPI, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI Configuration.

1. When DSPI executes last command word from a queue, 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_SR is set.
3. The setting of the EOQF flag disables serial transmission and reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is cleared to indicate the STOPPED state.
4. The DMA can continue to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the DMA Controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI_SR or by checking RFDF in the DSPI_SR after each read operation of the DSPI_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues

8. Flush TX FIFO by writing a ‘1’ to the CLR_TXF bit in the DSPI_MCR. Flush RX FIFO by writing a ‘1’ to the CLR_RXF bit in the DSPI_MCR.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to DSPI_TCR[TCNT].
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

28.5.2 Switching Master and Slave Mode

When changing modes in the DSPI, follow the steps below to guarantee proper operation.

1. Halt the DSPI by setting DSPI_MCR[HALT].
2. Clear the transmit and receive FIFOs by writing a 1 to the CLR_TXF and CLR_RXF bits in DSPI_MCR.
3. Set the appropriate mode in DSPI_MCR[MSTR] and enable the DSPI by clearing DSPI_MCR[HALT].

28.5.3 Baud Rate Settings

Table 357 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_CTAR registers. The values calculated assume a 100 MHz system frequency and the double baud rate DBR bit is clear.

Table 357. Baud Rate Values (bps)

		Baud Rate Divider Prescaler Values			
		2	**3**	**5**	**7**
Baud Rate Scaler Values	2	25.0M	16.7M	10.0M	7.14M
	4	12.5M	8.33M	5.00M	3.57M
	6	8.33M	5.56M	3.33M	2.38M
	8	6.25M	4.17M	2.50M	1.79M
	16	3.12M	2.08M	1.25M	893k
	32	1.56M	1.04M	625k	446k
	64	781k	521k	312k	223k
	128	391k	260k	156k	112k
	256	195k	130k	78.1k	55.8k
	512	97.7k	65.1k	39.1k	27.9k
	1024	48.8k	32.6k	19.5k	14.0k
	2048	24.4k	16.3k	9.77k	6.98k
	4096	12.2k	8.14k	4.88k	3.49k
	8192	6.10k	4.07k	2.44k	1.74k
	16384	3.05k	2.04k	1.22k	872
	32768	1.53k	1.02k	610	436

28.5.4 Delay Settings

Table 358 shows the values for the Delay after Transfer (t_{DT}) and CS to SCK Delay (T_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency.

This table does not apply for TSB Continuous mode.

Table 358. Delay Values

		Delay Prescaler Values			
		1	3	5	7
Delay Scaler Values	2	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

28.5.5 Calculation of FIFO Pointer Addresses

Complete visibility of the TX and RX FIFO contents is available 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). *Figure 409* illustrates the concept of first-in and last-in FIFO entries along with the FIFO Counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See *Section 28.4.2.4: Transmit First In First Out (TX FIFO) Buffering Mechanism*, and *Section 28.4.2.5: Receive First In First Out (RX FIFO) Buffering Mechanism*, for details on the FIFO operation.

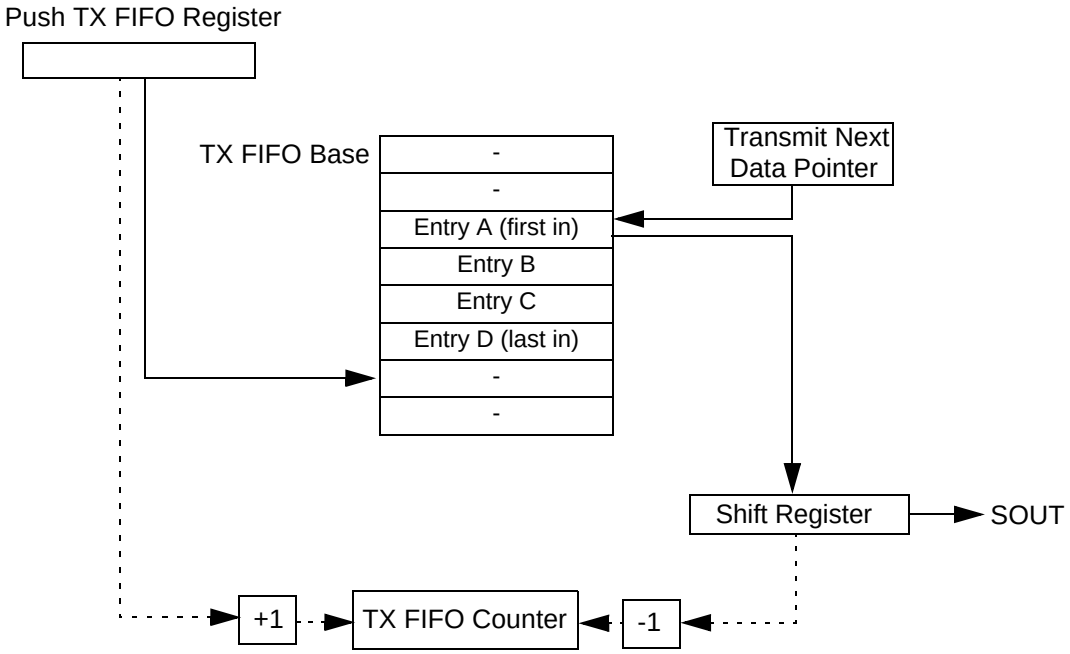


Figure 409. TX FIFO Pointers and Counter

28.5.5.1 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:

Equation 17 $$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 18

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times (\text{TXCTR} + \text{TXNXTPTR} - 1) \bmod (\text{TXFIFOdepth})$$

TX FIFO Base - Base address of TX FIFO

TXCTR - TX FIFO Counter

TXNXTPTR - Transmit Next Pointer

TX FIFO Depth - Transmit FIFO depth, implementation specific

28.5.5.2 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:

Equation 19

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 20

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times (\text{RXCTR} + \text{POPNXTPTR} - 1) \bmod (\text{RXFIFOdepth})$$

RX FIFO Base - Base address of RX FIFO

RXCTR - RX FIFO counter

POPNXTPTR - Pop Next Pointer

RX FIFO Depth - Receive FIFO depth, implementation specific

29 FlexRay Communication Controller (FLEXRAY)

29.1 Introduction

29.1.1 Reference

The following documents are referenced.

- FlexRay Communications System Protocol Specification, Version 2.1 Rev A[m]
- FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A

29.1.2 Glossary

This section provides a list of terms used in this chapter.

Table 359. List of Terms

Term	Definition
BCU	Buffer Control Unit. Handles message buffer access.
BMIF	Bus Master Interface. Provides master access to FlexRay memory area.
CC	Communication Controller
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in µT	The actual length of a cycle in µT for the ideal controller (+/- 0 ppm)
EBI	External Bus Interface
FlexRay Memory Area	Memory area to store the physical message buffer payload data, frame header, frame and slot status, and synchronization frame related tables.
System Memory	Memory that is contains the FlexRay Memory Area.
System Bus	Bus that connects the controller and System Memory
FSS	Frame Start Sequence
HIF	Host Interface. Provides host access to controller.
Host	The FlexRay CC host MCU
LUT	Look Up Table. Stores message buffer header index value.
LRAM	Look Up Table RAM. Module internal memory to store message buffer configuration data and data field offsets for individual message buffers and receive shadow buffers.
MB	Message Buffer
MBIDX	Message Buffer Index: the position of a header field entry within the header area. If the header area is accessed as an array, this is the same as the array index of the entry.
MBNum	Message Buffer Number: Position of message buffer configuration registers within the register map. For example, Message Buffer Number 5 corresponds to the MBCCS5 register.
MCU	Microcontroller Unit

m. The FlexRay Specifications have been developed for automotive applications.The FlexRay Specifications have been neither developed nor tested for non-automotive applications.

Table 359. List of Terms (continued)

Term	Definition
µT	Microtick
MT	Macrotick
MTS	Media Access Test Symbol
NIT	Network Idle Time
PE	Protocol Engine
POC	Protocol Operation Control. Each state of the POC is denoted by *POC:state*
Rx	Reception
SEQ	Sequencer Engine
TCU	Time Control Unit
Tx	Transmission
sync frame	null frame or message frame with *Sync Frame Indicator* set to 1
startup frame	null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 1
normal frame	null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 0
null frame	frame with *Null Frame Indicator* set to 0
message frame	frame with *Null Frame Indicator* set to 1

29.1.3 Color Coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

29.1.4 Overview

The CC is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The CC has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the CC with its surrounding modules is given in *Figure 410*.

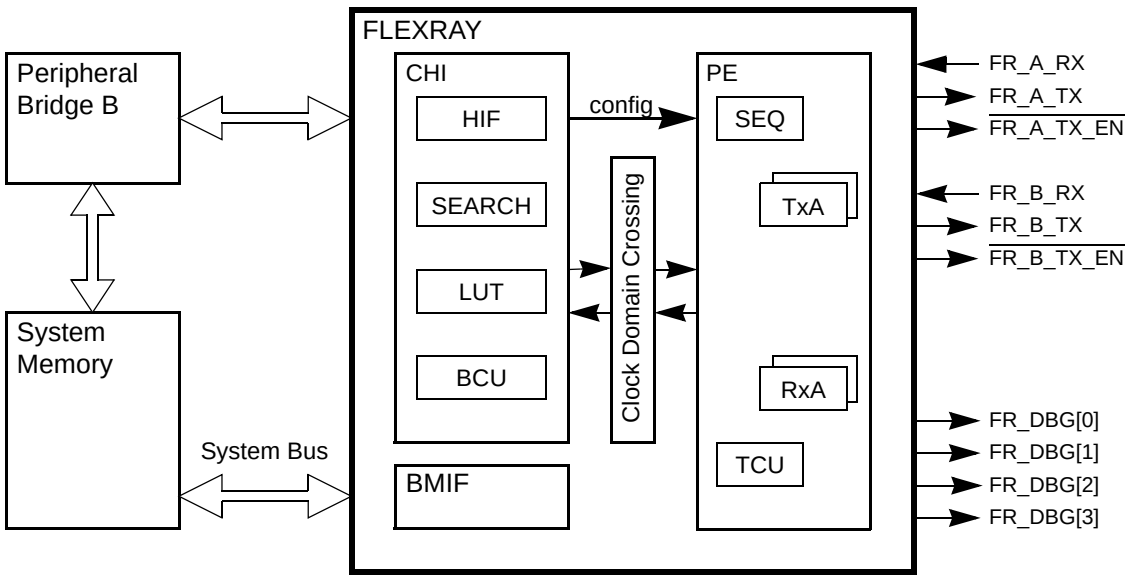


Figure 410. FLEXRAY Block Diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the flexray memory area.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The CC stores the frame header and payload data of frames received or of frames to be transmitted in the flexray memory area. The application accesses the flexray memory area to retrieve and provide the frames to be processed by the CC. In addition to the frame header and payload data, the CC stores the synchronization frame related tables in the flexray memory area for application processing.

The flexray memory area is located in the system memory of the MCU. The CC has access to the flexray memory area via its bus master interface (BMIF). The host provides the start address of the flexray memory area within the system memory by programming the *System Memory Base Address Register (FR_SYMBADR)*. All flexray memory area related offsets are stored in offset registers. The physical address pointer into the flexray memory area is calculated using the offset values the flexray memory base address.

Note: The CC does not provide a memory protection scheme for the flexray memory area.

29.1.5 Features

The CC provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A* compliant bus driver interface
- single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 128 configurable message buffers with
  - individual frame ID filtering
  - individual channel ID filtering
  - individual cycle counter filtering
- message buffer header, status and payload data stored in dedicated flexray memory area
  - allows for flexible and efficient message buffer implementation
  - consistent data access ensured by means of buffer locking scheme
  - application can lock multiple buffers at the same time
- size of message buffer payload data section configurable from 0 up to 254 bytes
- two independent message buffer segments with configurable size of payload data section
  - each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- zero padding for transmit message buffers in static segment
  - applied when the frame payload length exceeds the size of the message buffer data section
- transmit message buffers configurable with state/event semantics
- message buffers can be configured as
  - receive message buffer
  - transmit message buffer
- individual message buffer reconfiguration supported
  - means provided to safely disable individual message buffers
  - disabled message buffers can be reconfigured
- two independent receive FIFOs
  - one receive FIFO per channel
  - up to 255 entries for each FIFO
  - global frame ID filtering, based on both value/mask filters and range filters
  - global channel ID filtering
  - global message ID filtering for the dynamic segment
- 4 configurable slot error counters
- 4 dedicated slot status indicators
  - used to observe slots without using receive message buffers

- measured value indicators for the clock synchronization
  - internal synchronization frame ID and synchronization frame measurement tables can be copied into the flexray memory area
- fractional macroticks are supported for clock correction
- maskable interrupt sources provided via individual and combined interrupt lines
- 1 absolute timer
- 1 timer that can be configured to absolute or relative
- SECDED for protocol engine data ram
- SEDDED for chi lookup table ram

29.1.6 Modes of Operation

This section describes the basic operational power modes of the CC.

29.1.6.1 Disabled Mode

The CC enters the Disabled Mode during hard reset. The CC indicates that it is in the Disabled Mode by negating the module enable bit MEN in the *Module Configuration Register (FR_MCR)*.

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in *Section 29.5.2: Register Descriptions*.

The application configures the CC by accessing the configuration bits and fields in the *Module Configuration Register (FR_MCR)*.

29.1.6.1.1 Leave Disabled Mode

The CC leaves the Disabled Mode and enters the Normal Mode, when the application writes 1 to the module enable bit MEN in the *Module Configuration Register (FR_MCR)*

Note: When the CC was enabled, it cannot be disabled the later on.

29.1.6.2 Normal Mode

In this mode the CC is fully functional. The CC indicates that it is in Normal Mode by asserting the module enable bit MEN in the *Module Configuration Register (FR_MCR)*.

29.1.6.2.1 Enter Normal Mode

This mode is entered when the application requests the CC to leave the Disabled Mode. If the Normal Mode was entered by leaving the Disabled Mode, the application has to perform the protocol initialization described in *Section 29.7.2.2: Protocol Initialization*" to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the *Module Configuration Register (FR_MCR)*, the corresponding FlexRay bus driver ports are enabled and driven.

29.2 External Signal Description

This section lists and describes the CC signals, connected to external pins. These signals are summarized in *Table 360* and described in detail in *Section 29.2.1: Detailed Signal Descriptions*.

Note: The off chip signals FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ are available on each package option. The availability of the other off chip signals depends on the package option.

Table 360. External Signal Properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
FR_A_TX_EN	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
FR_B_TX_EN	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

29.2.1 Detailed Signal Descriptions

This section provides a detailed description of the CC signals, connected to external pins.

29.2.1.1 FR_A_RX — Receive Data Channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

29.2.1.2 FR_A_TX — Transmit Data Channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

29.2.1.3 $\overline{\text{FR_A_TX_EN}}$ — Transmit Enable Channel A

The $\overline{\text{FR_A_TX_EN}}$ signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel A.

29.2.1.4 FR_B_RX — Receive Data Channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

29.2.1.5 FR_B_TX — Transmit Data Channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

29.2.1.6 FR_B_TX_EN — Transmit Enable Channel B

The FR_B_TX_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel B.

29.2.1.7 FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0] — Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to *Section 29.6.16: Strobe Signal Support*.

29.3 Controller Host Interface Clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. There are two constraints for the minimum CHI clock frequency.

The first constraint corresponds to the number of utilized message buffers and is specified in *Section 29.7.5: Number of Usable Message Buffers*.

The second constraint corresponds to the value of the TIMEOUT field in the *System Memory Access Time-Out Register (FR_SYMATOR)* and is specified in *Section 29.7.1.1: Configure System Memory Access Time-Out Register (FR_SYMATOR)*.

29.4 Protocol Engine Clocking

The clock for the protocol engine can be generated by two sources. The first source is the external high speed crystal oscillator and the second source is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the *Module Configuration Register (FR_MCR)*.

29.4.1 Oscillator Clocking

If the protocol engine is clocked by the external crystal oscillator, a 40 MHz crystal or CMOS compatible clock must be connected to the oscillator pins. The crystal or clock must fulfill the requirements given by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

29.4.2 PLL Clocking

FlexRay protocol engine can be clocked by the internal PLL (see *Figure 30*). For PLL clocking, the input to FlexRay has to be 80Mhz. If the System clock output of PLL is selected as Protocol clock for FlexRay through AUX_CLK_MUX, then PLL needs to be programmed for 80Mhz frequency. If the PLL_PHI1_CLK output of PLL is selected, then PLL needs to be programmed such that VCO of the PLL is 480 MHz and PLL_PHI1_CLk is VCO/6 = 80Mhz.

29.5 Memory Map and Register Description

The CC occupies 8 KB (8192 bytes) of address space starting at the CCs base address defined by the memory map of the MCU.

29.5.1 Memory Map

The complete memory map of the CC is shown in *Table 361*. The addresses presented here are the offsets relative to the CC base address.

Table 361. FlexRay memory map

Base address: 0xFFFE_0000		
Address offset	**Register**	**Location**
0x0000	Module Version Register (FR_MVR)	*on page 807*
0x0002	Module Configuration Register (FR_MCR)	*on page 808*
0x0004	System Memory Base Address High Register (FR_SYMBADHR)	*on page 810*
0x0006	System Memory Base Address Low Register (FR_SYMBADLR)	*on page 810*
0x0008	Strobe Signal Control Register (FR_STBSCR)	*on page 811*
0x000A	Reserved	
0x000C	Message Buffer Data Size Register (FR_MBDSR)	*on page 812*
0x000E	Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)	*on page 813*
0x0010	PE DRAM Access Register (FR_PEDRAR)	*on page 813*
0x0012	PE DRAM Data Register (FR_PEDRDR)	*on page 814*
0x0014	Protocol Operation Control Register (FR_POCR)	*on page 814*
0x0016	Global Interrupt Flag and Enable Register (FR_GIFER)	*on page 817*
0x0018	Protocol Interrupt Flag Register 0 (FR_PIFR0)	*on page 819*
0x001A	Protocol Interrupt Flag Register 1 (FR_PIFR1)	*on page 821*
0x001C	Protocol Interrupt Enable Register 0 (FR_PIER0)	*on page 822*
0x001E	Protocol Interrupt Enable Register 1 (FR_PIER1)	*on page 824*
0x0020	CHI Error Flag Register (FR_CHIERFR)	*on page 825*
0x0022	Message Buffer Interrupt Vector Register (FR_MBIVEC)	*on page 827*
0x0024	Channel A Status Error Counter Register (FR_CASERCR)	*on page 828*
0x0026	Channel B Status Error Counter Register (FR_CBSERCR)	*on page 828*
0x0028	Protocol Status Register 0 (FR_PSR0)	*on page 829*
0x002A	Protocol Status Register 1 (FR_PSR1)	*on page 831*
0x002C	Protocol Status Register 2 (FR_PSR2)	*on page 832*
0x002E	Protocol Status Register 3 (FR_PSR3)	*on page 834*
0x0030	Macrotick Counter Register (FR_MTCTR)	*on page 835*
0x0032	Cycle Counter Register (FR_CYCTR)	*on page 836*
0x0034	Slot Counter Channel A Register (FR_SLTCTAR)	*on page 836*
0x0036	Slot Counter Channel B Register (FR_SLTCTBR)	*on page 836*
0x0038	Rate Correction Value Register (FR_RTCORVR)	*on page 837*
0x003A	Offset Correction Value Register (FR_OFCORVR)	*on page 837*

Table 361. FlexRay memory map (continued)

Base address: 0xFFFE_0000		
Address offset	**Register**	**Location**
0x003C	Combined Interrupt Flag Register (FR_CIFR)	on page 838
0x003E	System Memory Access Time-Out Register (FR_SYMATOR)	on page 839
0x0040	Sync Frame Counter Register (FR_SFCNTR)	on page 840
0x0042	Sync Frame Table Offset Register (FR_SFTOR)	on page 840
0x0044	Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)	on page 841
0x0046	Sync Frame ID Rejection Filter Register (FR_SFIDRFR)	on page 842
0x0048	Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)	on page 843
0x004A	Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)	on page 843
0x004C	Network Management Vector Register 0 (FR_NMVR0)	on page 844
0x004E	Network Management Vector Register 1 (FR_NMVR1)	on page 844
0x0050	Network Management Vector Register 2 (FR_NMVR2)	on page 844
0x0052	Network Management Vector Register 3 (FR_NMVR3)	on page 844
0x0054	Network Management Vector Register 4 (FR_NMVR4)	on page 844
0x0056	Network Management Vector Register 5 (FR_NMVR5)	on page 844
0x0058	Network Management Vector Length Register (FR_NMVLR)	on page 845
0x005A	Timer Configuration and Control Register (FR_TICCR)	on page 845
0x005C	Timer 1 Cycle Set Register (FR_TI1CYSR)	on page 846
0x005E	Timer 1 Macrotick Offset Register (FR_TI1MTOR)	on page 847
0x0060	Timer 2 Configuration Register 0 (FR_TI2CR0)	on page 847
0x0062	Timer 2 Configuration Register 1 (FR_TI2CR1)	on page 848
0x0064	Slot Status Selection Register (FR_SSSR)	on page 848
0x0066	Slot Status Counter Condition Register (FR_SSCCR)	on page 849
0x0068	Slot Status Register 0 (FR_SSR0)	on page 851
0x006A	Slot Status Register 1 (FR_SSR1)	on page 851
0x006C	Slot Status Register 2 (FR_SSR2)	on page 851
0x006E	Slot Status Register 3 (FR_SSR3)	on page 851
0x0070	Slot Status Register 4 (FR_SSR4)	on page 851
0x0072	Slot Status Register 5 (FR_SSR5)	on page 851
0x0074	Slot Status Register 6 (FR_SSR6)	on page 851
0x0076	Slot Status Register 7 (FR_SSR7)	on page 851
0x0078	Slot Status Counter Register 0 (FR_SSCR0)	on page 852
0x007A	Slot Status Counter Register 1 (FR_SSCR1)	on page 852
0x007C	Slot Status Counter Register 2 (FR_SSCR2)	on page 852

Table 361. FlexRay memory map (continued)

Base address: 0xFFFE_0000		
Address offset	**Register**	**Location**
0x007E	Slot Status Counter Register 3 (FR_SSCR3)	on page 852
0x0080	MTS A Configuration Register (FR_MTSACFR)	on page 853
0x0082	MTS B Configuration Register (MTSBCFR)	on page 853
0x0084	Receive Shadow Buffer Index Register (FR_RSBIR)	on page 854
0x0086	Receive FIFO Watermark and Selection Register (FR_RFWMSR)	on page 856
0x0088	Receive FIFO Start Index Register (FR_RFSIR)	on page 857
0x008A	Receive FIFO Depth and Size Register (RFDSR)	on page 857
0x008C	Receive FIFO A Read Index Register (FR_RFARIR)	on page 858
0x008E	Receive FIFO B Read Index Register (FR_RFBRIR)	on page 858
0x0090	Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)	on page 859
0x0092	Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)	on page 860
0x0094	Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)	on page 860
0x0096	Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)	on page 860
0x0098	Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)	on page 861
0x009A	Receive FIFO Range Filter Control Register (FR_RFRFCTR)	on page 861
0x009C	Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)	on page 862
0x009E	Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)	on page 863
0x00A0 ... 0x00DC	Protocol Configuration Register 0 (FR_PCR0) ... Protocol Configuration Register 30 (FR_PCR30)	on page 865 ... on page 872
0x00DE ... 0x00E4	Reserved	
0x00E6	Receive FIFO Start Data Offset Register (FR_RFSDOR)	on page 855
0x00E8	Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)	on page 855
0x00EA	Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)	on page 855
0x00EC	Receive FIFO Periodic Timer Register (FR_RFPTR)	on page 856
0x00EE	Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)	on page 859
0x00F0	ECC Error Interrupt Flag and Enable Register (FR_EEIFER)	on page 872
0x00F2	ECC Error Report and Injection Control Register (FR_EERICR)	on page 874
0x00F4	ECC Error Report Address Register (FR_EERAR)	on page 875
0x00F6	ECC Error Report Data Register (FR_EERDR)	on page 876

Table 361. FlexRay memory map (continued)

Base address: 0xFFFE_0000		
Address offset	**Register**	**Location**
0x00F8	ECC Error Report Code Register (FR_EERCR)	*on page 877*
0x00FA	ECC Error Injection Address Register (FR_EEIAR)	*on page 878*
0x00FC	ECC Error Injection Data Register (FR_EEIDR)	*on page 878*
0x00FE	ECC Error Injection Code Register (FR_EEICR)	*on page 879*
0x0100 ... 0x07FE	Reserved	
0x0800	Message Buffer Configuration, Control, Status Register 0 (FR_MBCCSR0)	*on page 879*
0x0802	Message Buffer Cycle Counter Filter Register 0 (FR_MBCCFR0)	*on page 881*
0x0804	Message Buffer Frame ID Register 0 (FR_MBFIDR0)	*on page 882*
0x0806	Message Buffer Index Register 0 (FR_MBIDXR0)	*on page 882*
...	...	...
0x0BF8	Message Buffer Configuration, Control, Status Register 127 (FR_MBCCSR127)	*on page 879*
0x0BFA	Message Buffer Cycle Counter Filter Register 127 (FR_MBCCFR127)	*on page 881*
0x0BFC	Message Buffer Frame ID Register 127 (FR_MBFIDR127)	*on page 882*
0x0BFE	Message Buffer Index Register 127 (FR_MBIDXR127)	*on page 882*
0x0C00 ... 0x0FFF	Reserved	
0x1000 ... 0x1106	Message Buffer Data Field Offset Register 0 (FR_MBDOR0) ... Message Buffer Data Field Offset Register 131 (FR_MBDOR131)	*on page 883*
0x1108 ... 0x1112	LRAM ECC Error Test Register 0 (FR_LEETR0) ... LRAM ECC Error Test Register 5 (FR_LEETR5)	*on page 883*
0x1114 ... 0x1FFE	Reserved	

29.5.2 Register Descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 362 provides a key for the register figures and register tables.

Table 362. Register Access Conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
–	Not defined after reset and not affected by reset.

29.5.2.1 Register Reset

All registers except the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn), Message Buffer Frame ID Registers (FR_MBFIDRn), and Message Buffer Index Registers (FR_MBIDXRn) are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in *Table 363*.

Table 363. Additional Register Reset Conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command “0101” to the POCCMD field in the *Protocol Operation Control Register (FR_POCR)*.
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit FR_MBCCSRn[EDT] while the message buffer is enabled (FR_MBCCSRn[EDS] = 1) and the CC grants the disable to the application by clearing the FR_MBCCSRn[EDS] bit.

29.5.2.2 Register Write Access

This section describes the write access restriction terms that apply to all registers.

29.5.2.2.1 Register Write Access Restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in *Table 364*. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the

register or field can be written to if at least one of the conditions is fulfilled.The condition term [A and B] indicates that the register or field can be written to if both conditions are fulfilled.

Table 364. Register Write Access Restrictions

Condition	Indication	Description
Any Time	-	No write access restriction.
Disabled Mode	FR_MCR[MEN] = 0	Write access only when CC is in Disabled Mode.
Normal Mode	FR_MCR[MEN] = 1	Write access only when CC is in Normal Mode.
POC:config	FR_PSR0[PROTSTATE] = *POC:config*	Write access only when Protocol is in the *POC:config* state.
MB_DIS	FR_MBCCSR[EDS] = 0	Write access only when related Message Buffer is disabled.
MB_LCK	FR_MBCCSRn[LCKS] = 1	Write access only when related Message Buffer is locked.
IDL	FR_EEIRICR[BSY] = 0	Write access only when ECC configuration is idle

29.5.2.2.2 Register Write Access Requirements

All registers can be accessed with 8-bit, 16-bit and 32-bit wide operations.

For some of the registers, at least a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected. If an 8-bit wide write access is performed to any of these registers, this access is ignored without notification.

29.5.2.2.3 Internal Register Access

The following memory mapped registers are used to access multiple internal registers.

- *Strobe Signal Control Register (FR_STBSCR)*
- *Slot Status Selection Register (FR_SSSR)*
- *Slot Status Counter Condition Register (FR_SSCCR)*
- *Receive Shadow Buffer Index Register (FR_RSBIR)*

Each of these memory mapped registers provides a SEL field and a WMD bit. The SEL field is used to select the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

29.5.2.3 Module Version Register (FR_MVR)

Base + 0x0000

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

Figure 411. Module Version Register (FR_MVR)

This register provides the CC version number. The module version number is derived from the CHI version number and the PE version number.

Table 365. FR_MVR Field Descriptions

Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the controller host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

29.5.2.4 Module Configuration Register (FR_MCR)

Base + 0x0002 Write: MEN, SBFF, SCM, CHB, CHA, ECCE, FUM, FAM, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	ECCE	R*	FUM	FAM	0	CLK SEL	BITRATE			0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 412. Module Configuration Register (FR_MCR)

This register defines the global configuration of the CC.

Table 366. FR_MCR Field Descriptions

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the CC is in the Disabled Mode. The application requests the CC to leave the Disabled Mode by writing 1 to this bit Before leaving the Disabled Mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, BITRATE values. For details see *Section 29.1.6: Modes of Operation*. 0 Write: ignored, CC disable not possible Read: CC disabled 1 Write: enable CC Read: CC enabled **Note:** If the CC is enabled it can not be disabled.
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the CC in case of a system bus failure. 0 Continue normal operation 1 Transition to freeze mode
SCM	**Single Channel Device Mode** — This control bit defines the channel device mode of the CC as described in *Section 29.6.10: Channel Device Modes*. 0 CC works in dual channel device mode 1 CC works in single channel device mode
CHB CHA	**Channel Enable** — protocol related parameter: *pChannels* The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given *Table 367*.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see *Section 29.6.15: Sync Frame Filtering*. 0 Synchronization frame filtering disabled 1 Synchronization frame filtering enabled

Table 366. FR_MCR Field Descriptions (continued)

Field	Description
ECCE	**ECC Functionality Enable** — This bit controls the ecc memory error detection functionality. For details see *Section 29.6.24: Memory Content Error Detection*. 0 ECC functionality (injection, detection, reporting, response) disabled 1 ECC functionality enabled
FUM	**FIFO Update Mode** — This bit controls the FIFO update behavior when the interrupt flags FR_GIFER[FAFAIF] and FR_GIFER[FAFBIF] are written by the application (see *Section 29.6.9.8: FIFO Update*) 0 FIFOA/FIFOB is updated on writing 1 to FR_GIFER[FAFAIF] /FR_GIFER[FAFBIF] 1 FIFOA/FIFOB) is *not* updated on writing 1 to FR_GIFER[FAFAIF]/FR_GIFER[FAFBIF]
FAM	**FIFO Address Mode** — This bit controls the location of the system memory base address for the FIFOs. (see *Section 29.6.9.2: FIFO Configuration*) 0 FIFO Base Address located in *System Memory Base Address Register (FR_SYMBADR)* 1 FIFO Base Address located in *Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)*
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator. 1 PE clock source is generated by on-chip PLL. **Note:** Before changing the clock source, the FlexRay module should be disabled to prevent any communication glitches.
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the FlexRay Bus Bit Rate. 000 10.0 Mbit/sec 001 5.0 Mbit/sec 010 2.5 Mbit/sec 011 8.0 Mbit/sec 100 reserved 101 reserved 110 reserved 111 reserved

Table 367. FlexRay Channel Selection

SCM	CHB	CHA	Description
Dual Channel Device Modes			
0	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by CC ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by CC ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by CC - connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by CC - connected to FlexRay channel B
Single Channel Device Mode			

Table 367. FlexRay Channel Selection

SCM	CHB	CHA	Description
1	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	1	reserved

29.5.2.5 System Memory Base Address Register (FR_SYMBADR)

Base + 0x0004 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SMBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 413. System Memory Base Address High Register (FR_SYMBADHR)

Base + 0x0006 Write: Disabled Mode

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

Figure 414. System Memory Base Address Low Register (FR_SYMBADLR)

Note: The system memory base address must be set before the CC is enabled.

The system memory base address registers define the base address of the flexray memory area within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 368. FR_SYMBADR Field Descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the individual message buffers and sync frame table. This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defines as a byte address.

29.5.2.6 Strobe Signal Control Register (FR_STBSCR)

Base + 0x0008 16-bit write access required Write: Anytime

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

Figure 415. Strobe Signal Control Register (FR_STBSCR)

This register is used to assign the individual protocol timing related strobe signals given in *Table 370* to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to *Section 29.6.16: Strobe Signal Support*.

Note: In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 369. FR_STBSCR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in *Table 370* to be enabled or disabled and assigned to one of the four strobe ports given in *Table 370*.
ENB	**Strobe Signal Enable** — The control bit is used to enable and to disable the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 assign selected signal to FR_DBG[0] 01 assign selected signal to FR_DBG[1] 10 assign selected signal to FR_DBG[2] 11 assign selected signal to FR_DBG[3]

Table 370. Strobe Signal Mapping

SEL		Description	Channel	Type	Offset (1)	Reference
dec	hex					
0	0x0	arm	-	value	+1	MT start
1	0x1	mt	-	value	+1	MT start
2	0x2	cycle start	-	pulse	0	MT start
3	0x3	minislot start	-	pulse	0	MT start

Table 370. Strobe Signal Mapping

SEL		Description	Channel	Type	Offset (1)	Reference
dec	hex					
4	0x4	slot start	A	pulse	0	MT start
5	0x5		B			
6	0x6	receive data after glitch filtering	A	value	+4	FR_A_RX
7	0x7		B			FR_B_RX
8	0x8	channel idle indicator	A	level	+5	FR_A_RX
9	0x9		B			FR_B_RX
10	0xA	syntax error detected	A	pulse	+4	FR_A_RX
11	0xB		B			FR_B_RX
12	0xC	content error detected	A	level	+4	FR_A_RX
13	0xD		B			FR_B_RX
14	0xE	receive FIFO almost-full interrupt signals	A	value	n.a.	RX FIFO A Almost Full Interrupt
15	0xF		B			RX FIFO B Almost Full Interrupt

1. Given in PE clock cycles

29.5.2.7 Message Buffer Data Size Register (FR_MBDSR)

Base + 0x000C Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MBSEG2DS							0	MBSEG1DS						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 416. Message Buffer Data Size Register (FR_MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The CC provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 371. FR_MBDSR Field Descriptions

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

29.5.2.8 Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

Base + 0x000E Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	LAST_MB_SEG1							0	LAST_MB_UTIL						
W																
Reset	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

Figure 417. Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

This register is used to define the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 372. FR_MBSSUTR Field Descriptions

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn with n <= LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG1+1 individual message buffers. **Note:** The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn with LAST_MB_SEG1 < n < 128. **Note:** If LAST_MB_SEG1 = 127 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number n <= LAST_MB_UTIL. **Note:** If LAST_MB_UTIL=LAST_MB_SEG1 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

29.5.2.9 PE DRAM Access Register (FR_PEDRAR)

Base + 0x0010 16-bit write access required Write: Normal Mode

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

Figure 418. PE DRAM Access Register (FR_PEDRAR)

This register is used to trigger write and read operations on the PE data memory (PE DRAM). These operations are used for memory error injection and memory error observation.

Each write access to this registers initiates a read or write operation on the PE DRAM. The access done status bit DAD is cleared after the write access and is set if the PE DRAM access has been finished.

In case of an PE DRAM write access, the data provided in *PE DRAM Data Register (FR_PEDRDR)* are written into the PE DRAM, read back from the PE DRAM and are stored into the *PE DRAM Data Register (FR_PEDRDR)*.

In case of an PE DRAM read access, the requested data are read from PE DRAM and stored into the *PE DRAM Data Register (FR_PEDRDR)*.

For a detailed description refer to *Section 29.6.24: Memory Content Error Detection*

Table 373. FR_PEDRAR Field Descriptions

Field	Description
INST	**PE DRAM Access Instruction** — This field defines the operation to be executed on the PE DRAM. 0011 PE DRAM write: Write FR_PEDRDR[DATA] to PE DRAM address ADDR (16 bit) 0101 PE DRAM read: Read Data from PE DRAM address ADDR (16 bit) into FR_PEDRDR[DATA] other reserved
ADDR	**PE DRAM Access Address** — This field defines the address in the PE DRAM to be written to or read from.
DAD	**PE DRAM Access Done** — This status bit is cleared when the application has written to this register and is set when the PE DRAM access has finished. 0 PE DRAM access running 1 PE DRAM access done

29.5.2.10 PE DRAM Data Register (FR_PEDRDR)

Base + 0x0012 16-bit write access required Write: Normal Mode

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

Figure 419. PE DRAM Data Register (FR_PEDRDR)

This register provides the data to be written to or read from the PE DRAM by the access initiated by write access to the PE DRAM Access Register (FR_PEDRAR).

29.5.2.11 Protocol Operation Control Register (FR_POCR)

Base + 0x0014 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 420. Protocol Operation Control Register (FR_POCR)

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see *Section 29.7.6: Protocol Control Command Execution*.

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to *Section 29.6.11: External Clock Synchronization*.

Table 374. FR_POCR Field Descriptions

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.

Table 374. FR_POCR Field Descriptions

Field	Description
EOC_AP	**External Offset Correction Application** — This field is used to trigger the application of the external offset correction value defined in the *Protocol Configuration Register 29 (FR_PCR29)*. 00 do not apply external offset correction value 01 reserved 10 subtract external offset correction value 11 add external offset correction value
ERC_AP	**External Rate Correction Application** — This field is used to trigger application of the external rate correction value defined in the *Protocol Configuration Register 21 (FR_PCR21)* 00 do not apply external rate correction value 01 reserved 10 subtract external rate correction value 11 add external rate correction value
BSY WMC	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The CC sets this status bit when the application has issued a protocol control command via the POCCMD field. The CC clears this status bit when protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the CC ignores this command, sets the protocol command ignored error flag PCMI_EF in the *CHI Error Flag Register (FR_CHIERFR)*, and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command. **Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The CC sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed(1) transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

1. Delayed means on completion of current communication cycle.

29.5.2.12 Global Interrupt Flag and Enable Register (FR_GIFER)

Base + 0x0016 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUPIF	FAFBIF	FAFAIF	RBIF	TBIF	MIE	PRIE	CHIE	WUPIE	FAFBIE	FAFAIE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 421. Global Interrupt Flag and Enable Register (FR_GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is depicted in *Figure 566*. For more details on interrupt generation, see *Section 29.6.20: Interrupt Support*. These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 375. FR_GIFER Field Descriptions

Field	Description
MIF	**Module Interrupt Flag** — This interrupt flag is set if at least one of the other interrupt flags in this register and the related interrupt enable bit are set. 0 No interrupt flag and related interrupt enable bit are set 1 At least one of the other interrupt flags in this register and the related interrupt bit are set.
PRIF	**Protocol Interrupt Flag** — This interrupt flag is set if at least one of the individual flags in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* and *Protocol Interrupt Flag Register 1 (FR_PIFR1)* and the related interrupt enable bit are set. 0 No individual protocol interrupt flag and related interrupt enable bit are set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable bit are set.
CHIF	**CHI Interrupt Flag** — This interrupt flag is set if at least one of the error flags in the *CHI Error Flag Register (FR_CHIERFR)* and the chi error interrupt enable bit FR_GIFER[CHIE] are set. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled. 1 At least one CHI error flag and the chi error interrupt enable are is set.
WUPIF	**Wakeup Interrupt Flag** — This interrupt flag is set when the CC has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the *Protocol Status Register 3 (FR_PSR3)*. 0 No Wakeup symbol received on FlexRay bus 1 Wakeup symbol received on FlexRay bus
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — This interrupt flag is set when one of the following events occurs a) the current number of FIFO B entries is equal to or greater than the watermark defined by the WM field in the *Receive FIFO Watermark and Selection Register (FR_RFWMSR)*, and the CC writes a received message into the FIFO B, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by *Receive FIFO Periodic Timer Register (FR_RFPTR)* expires. 0 no such event 1 FIFO B almost full event has occurred

Table 375. FR_GIFER Field Descriptions (continued)

Field	Description
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — This interrupt flag is set when one of the following events occurs a) the current number of FIFO A entries is equal to or greater than the watermark defined by the WM field in the *Receive FIFO Watermark and Selection Register (FR_RFWMSR)*, and the CC writes a received message into the FIFO A, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by *Receive FIFO Periodic Timer Register (FR_RFPTR)* expires. 0 no such event 1 FIFO A almost full event has occurred
RBIF	**Receive Message Buffer Interrupt Flag** — This interrupt flag is set if for at least one of the individual receive message buffers (FR_MBCCSRn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* are asserted. The application can not clear this interrupt flag directly, instead it is cleared by the CC when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the related interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag set. 1 At least one individual receive message buffer has the MBIF and MBIE flag set.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual message buffers (FR_MBCCSRn[MTD] = 1) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* are equal to 1. The application can not clear this interrupt flag directly, instead, this interrupt flag is cleared by the CC when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the application has cleared the related interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag set. 1 At least one individual transmit message buffer has the MBIF and MBIE flag set.
MIE	**Module Interrupt Enable** — This bit controls if the Module Interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	**Protocol Interrupt Enable** — This bit controls if the Protocol Interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	**CHI Interrupt Enable** — This bit controls if the CHI Interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
WUPIE	**Wakeup Interrupt Enable** — This bit controls if the Wakeup Interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Table 375. FR_GIFER Field Descriptions (continued)

Field	Description
FAFBIE	**Receive FIFO Channel B Almost Full Interrupt Enable** — This bit controls if the RX FIFO B Almost Full Interrupt line is asserted when the FAFBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FAFAIE	**Receive FIFO Channel A Almost Full Interrupt Enable** — This bit controls if the RX FIFO A Almost Full Interrupt line is asserted when the FAFAIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
RBIE	**Receive Message Buffer Interrupt Enable** — This bit controls if the Receive Message Buffer Interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
TBIE	**Transmit Message Buffer Interrupt Enable** — This bit controls if the Transmit Message Buffer Interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

29.5.2.13 Protocol Interrupt Flag Register 0 (FR_PIFR0)

Base + 0x0018 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 422. Protocol Interrupt Flag Register 0 (FR_PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 376. FR_PIFR0 Field Descriptions

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: 1) *pLatestTx* violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the *Protocol Configuration Register 14 (FR_PCR14)* and *Protocol Configuration Register 15 (FR_PCR15)* when the CONFIG_COMPLETE command was sent by the application via the *Protocol Operation Control Register (FR_POCR)*. If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the *Protocol Configuration Register 3 (FR_PCR3)*. 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event 1 Insufficient number of measurements for rate correction detected
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the *offset_coorection_out* field in the *Protocol Configuration Register 9 (FR_PCR9)* and the *rate_correction_out* field in the *Protocol Configuration Register 14 (FR_PCR14)*. 0 No such event. 1 Offset or rate correction limit reached.
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the *Protocol Configuration Register 30 (FR_PCR30)*. 0 No such event. 1 More than *node_sync_max* sync frames detected. **Note:** Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.

Table 376. FR_PIFR0 Field Descriptions (continued)

Field	Description
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

29.5.2.14 Protocol Interrupt Flag Register 1 (FR_PIFR1)

Base + 0x001A — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 423. Protocol Interrupt Flag Register 1 (FR_PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 377. FR_PIFR1 Field Descriptions

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the *Protocol Status Register 0 (FR_PSR0)* is changed by the CC. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the *Protocol Operation Control Register (FR_POCR)*, and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see *Section 29.7.6: Protocol Control Command Execution*". 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the CC has detected a communication failure between the PE and the CHI. 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the *Protocol Status Register 0 (FR_PSR0)* has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding *Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)* is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the flexray memory area for the even cycle. 0 No such event. 1 Sync frame measurement table written
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the flexray memory area for the odd cycle. 0 No such event. 1 Sync frame measurement table written

29.5.2.15 Protocol Interrupt Enable Register 0 (FR_PIER0)

Base + 0x001C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 424. Protocol Interrupt Enable Register 0 (FR_PIER0)

This register defines whether or not the individual interrupt flags defined in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* can generate a protocol interrupt request.

Table 378. FR_PIER0 Field Descriptions

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVB_IE	**Transmission across boundary on channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 378. FR_PIER0 Field Descriptions (continued)

Field	Description
TBVA_IE	**Transmission across boundary on channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

29.5.2.16 Protocol Interrupt Enable Register 1 (FR_PIER1)

Base + 0x001E — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PECF_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 425. Protocol Interrupt Enable Register 1 (FR_PIER1)

This register defines whether or not the individual interrupt flags defined in *Protocol Interrupt Flag Register 1 (FR_PIFR1)* can generate a protocol interrupt request.

Table 379. FR_PIER1 Field Descriptions

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 379. FR_PIER1 Field Descriptions (continued)

Field	Description
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

29.5.2.17 CHI Error Flag Register (FR_CHIERFR)

Base + 0x0020 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	0	SBCF_EF	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 426. CHI Error Flag Register (FR_CHIERFR)

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the *Global Interrupt Flag and Enable Register (FR_GIFER)*.

Table 380. FR_CHIERFR Field Descriptions

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event 1 Frame lost on channel B detected
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error 1 Frame lost on channel A detected
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the *Protocol Operation Control Register (FR_POCR)* while the BSY flag is equal to 1. In this case the command is ignored by the CC and is lost. 0 No such error 1 POC command ignored
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 FIFO overrun on channel B has been detected
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 FIFO overrun on channel B has been detected
MBS_EF	**Message Buffer Search Error Flag** — This flag is set if at least one of the following events occurs: a) The message buffer search engine is still running while the next search must be started due to the FlexRay protocol timing. b) A message buffer index greater than 131 is detected in the FR_MBIDXR[MBIDX] field of an found message buffer or in one of the FR_RSBIR[RSBIDX] fields. Refer to *Section 29.6.7.4: Message Buffer Search Error* for details. 0 No such event 1 Search engine active while search start appears or illegal message buffer index detected
MBU_EF	Message Buffer Utilization Error Flag — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*. If the application writes to a FR_MBCCSRn register with n > LAST_MB_UTIL, the CC ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the *CHI Error Flag Register (FR_CHIERFR)*. 0 No such event 1 Non-utilized message buffer enabled
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the CC due to internal operations. In that case, the CC does not grant the lock to the application. The application must issue the lock request again. 0 No such error 1 Lock error detected

Table 380. FR_CHIERFR Field Descriptions (continued)

Field	Description
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see *Section 29.6.19.1.2: System Bus Access Timeout*). 0 No such event 1 System bus access not finished in time
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred 1 Frame ID error occurred
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the *Protocol Configuration Register 24 (FR_PCR24)*. 0 No such error occurred 1 Dynamic payload length error occurred
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the *Protocol Configuration Register 19 (FR_PCR19)*. 0 No such error occurred 1 Static payload length error occurred
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the *Network Management Vector Length Register (FR_NMVLR)*. In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred 1 Network management length error occurred
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see *Network Management Vector Registers (FR_NMVR0–FR_NMVR5)*) are not updated. 0 No such error occurred 1 Network management frame error occurred
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the CC (see *Section 29.6.19.1.1: System Bus Illegal Address Access*). 0 No such event 1 Illegal system bus address accessed

29.5.2.18 Message Buffer Interrupt Vector Register (FR_MBIVEC)

Base + 0x0022

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

Figure 427. Message Buffer Interrupt Vector Register (FR_MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 381. FR_MBIVEC Field Descriptions

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

29.5.2.19 Channel A Status Error Counter Register (FR_CASERCR)

Base + 0x0024 Additional Reset: RUN Command

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

Figure 428. Channel A Status Error Counter Register (FR_CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see *Section 29.6.18: Slot Status Monitoring*.

Table 382. FR_CASERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

29.5.2.20 Channel B Status Error Counter Register (FR_CBSERCR)

Base + 0x0026 Additional Reset: RUN Command

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

Figure 429. Channel B Status Error Counter Register (FR_CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see *Section 29.6.18: Slot Status Monitoring*.

Table 383. FR_CBSERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

29.5.2.21 Protocol Status Register 0 (FR_PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 430. Protocol Status Register 0 (FR_PSR0)

This register provides information about the current protocol status.

Table 384. FR_PSR0 Field Descriptions

Field	Description
ERRMODE	**Error Mode** — protocol related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 reserved
SLOTMODE	**Slot Mode** — protocol related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 reserved
PROTSTATE	**Protocol State** — protocol related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*
STARTUP STATE	**Startup State** — protocol related variable: *vPOC!StartupState.* This field indicates the current sub-state of the startup procedure. 0000 reserved 0001 reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integrationi listen* 0110 reserved 0111 *POC:initialize schedule* 1000 reserved 1001 reserved 1010 *POC:coldstart consistency check* 1011 reserved 1100 reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — protocol related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 reserved

29.5.2.22 Protocol Status Register 1 (FR_PSR1)

Base + 0x002A Additional Reset: CSAA, CSP, CPN: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 431. Protocol Status Register 1 (FR_PSR1)

Table 385. FR_PSR1 Field Descriptions

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — protocol related event: 'set coldstart abort indicator in CHI' This flag is set when the CC has aborted a cold start attempt. 0 No such event 1 Cold start attempt aborted
CSP	**Leading Cold Start Path** — This status bit is set when the CC has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event 1 *POC:normal active* reached from *POC:startup* state via leading cold start path
REMCSAT	**Remaining Coldstart Attempts** — protocol related variable: *vRemainingColdstartAttempts* This field provides the number of remaining cold start attempts that the CC will execute.
CPN	**Leading Cold Start Path Noise** — protocol related variable: *vPOC!ColdstartNoise* This status bit is set if the CC has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the CC was starting up the cluster. 0 No such event 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path
HHR	**Host Halt Request Pending** — protocol related variable: *vPOC!CHIHaltRequest* This status bit is set when CC receives the HALT command from the application via the *Protocol Operation Control Register (FR_POCR)*. The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 HALT command received
FRZ	**Freeze Occurred** — protocol related variable: *vPOC!Freeze* This status bit is set when the CC has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 Immediate halt due to FREEZE or internal error condition
APTAC	**Allow Passive to Active Counter** — protocol related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the *Protocol Configuration Register 12 (FR_PCR12)*.

29.5.2.23 Protocol Status Register 2 (FR_PSR2)

Base + 0x002C Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 432. Protocol Status Register 2 (FR_PSR2)

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the CC after the end of the NIT and before the end of the first slot of the next communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the CC after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the CC after the end of the static segment and before the end of the current communication cycle.

Table 386. FR_PSR2 Field Descriptions

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEB	**NIT Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event 1 Syntax error detected
STCB	**Symbol Window Transmit Conflict on Channel B** — protocol related variable: *vSS!TxConflict* for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event 1 Transmission conflict detected
SBVB	**Symbol Window Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEB	**Symbol Window Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event 1 Syntax error detected

Table 386. FR_PSR2 Field Descriptions (continued)

Field	Description
MTB	**Media Access Test Symbol MTS Received on Channel B** — protocol related variable: *vSS!ValidMTS* for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event 1 MTS symbol received
NBVA	**NIT Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEA	**NIT Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event 1 Syntax error detected
STCA	**Symbol Window Transmit Conflict on Channel A** — protocol related variable: *vSS!TxConflict* for symbol window on channel A This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event 1 Transmission conflict detected
SBVA	**Symbol Window Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEA	**Symbol Window Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event 1 Syntax error detected
MTA	**Media Access Test Symbol MTS Received on Channel A** — protocol related variable: *vSS!ValidMTS* for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received 0 No such event
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — protocol related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the *Protocol Configuration Register 8 (FR_PCR8)*. The CC resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

29.5.2.24 Protocol Status Register 3 (FR_PSR3)

Base + 0x002E — Additional Reset: RUN Command — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 433. Protocol Status Register 3 (FR_PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 387. FR_PSR3 Field Descriptions

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received 1 Wakeup symbol received
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received 0 No syntactically valid frames received
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received 1 Wakeup symbol received

Table 387. FR_PSR3 Field Descriptions (continued)

Field	Description
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received 1 At least one syntactically valid frame received

29.5.2.25 Macrotick Counter Register (FR_MTCTR)

Base + 0x0030

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

Figure 434. Macrotick Counter Register (FR_MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 388. FR_MTCTR Field Descriptions

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

29.5.2.26 Cycle Counter Register (FR_CYCTR)

Base + 0x0032

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

Figure 435. Cycle Counter Register (FR_CYCTR)

This register provides the number of the current communication cycle.

Table 389. FR_CYCTR Field Descriptions

Field	Description
CYCCNT	**Cycle Counter** — protocol related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

29.5.2.27 Slot Counter Channel A Register (FR_SLTCTAR)

Base + 0x0034

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

Figure 436. Slot Counter Channel A Register (FR_SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 390. FR_SLTCTAR Field Descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — protocol related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

29.5.2.28 Slot Counter Channel B Register (FR_SLTCTBR)

Base + 0x0036

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

Figure 437. Slot Counter Channel B Register (FR_SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 391. FR_SLTCTBR Field Descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel B** — protocol related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

29.5.2.29 Rate Correction Value Register (FR_RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

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

Figure 438. Rate Correction Value Register (FR_RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT of each odd numbered communication cycle.

Table 392. FR_RTCORVR Field Descriptions

Field	Description
RATECORR	**Rate Correction Value** — protocol related variable: *vRateCorrection* (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the *Protocol Configuration Register 13 (FR_PCR13)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)*. **Note:** If the CC was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

29.5.2.30 Offset Correction Value Register (FR_OFCORVR)

Base + 0x003A Additional Reset: RUN Command

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

Figure 439. Offset Correction Value Register (FR_OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT.

Table 393. FR_OFCORVR Field Descriptions

Field	Description
OFFSET-CORR	**Offset Correction Value** — protocol related variable: *vOffsetCorrection* (before value limitation and external offset correction) This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the *Protocol Configuration Register 29 (FR_PCR29)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)*. **Note:** If the CC was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

29.5.2.31 Combined Interrupt Flag Register (FR_CIFR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUPIF	FAFBIF	FAFAIF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 440. Combined Interrupt Flag Register (FR_CIFR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is depicted in *Figure 568*. The individual interrupt flags WUPIF, FAFBIF, and FAFAIF are copies of corresponding flags in the *Global Interrupt Flag and Enable Register (FR_GIFER)* and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the *Global Interrupt Flag and Enable Register (FR_GIFER)*.

Note: The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Global Interrupt Flag and Enable Register (FR_GIFER).

Table 394. FR_CIFR Field Descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted 1 At least one interrupt source has its interrupt flag asserted
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* or *Protocol Interrupt Flag Register 1 (FR_PIFR1)* is equal to 1. 0 All individual protocol interrupt flags are equal to 0 1 At least one of the individual protocol interrupt flags is equal to 1
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *CHI Error Flag Register (FR_CHIERFR)* is equal to 1. 0 All CHI error flags are equal to 0 1 At least one CHI error flag is equal to 1
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as FR_GIFER[WUPIF]
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFBIF]
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFAIF]
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (FR_MBCCSRn[MTD] = 0) the interrupt flag MBIF in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual transmit message buffers (FR_MBCCSRn[MTD] = 1) the interrupt flag MBIF in the corresponding *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

29.5.2.32 System Memory Access Time-Out Register (FR_SYMATOR)

Base + 0x003E — Write: Disabled Mode

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

Figure 441. System Memory Access Time-Out Register (FR_SYMATOR)

Table 395. FR_SYMATOR Field Descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines when a system bus access timout is detected. For a detailed description see *Section 29.7.1.1: Configure System Memory Access Time-Out Register (FR_SYMATOR)* and *Section 29.6.19.1.2: System Bus Access Timeout*.

29.5.2.33 Sync Frame Counter Register (FR_SFCNTR)

Base + 0x0040 Additional Reset: RUN Command

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

Figure 442. Sync Frame Counter Register (FR_SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

Note: If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the CC will not update the fields SFEVB and SFEVA.

Note: If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the CC will not update the values SFODB and SFODA.

Table 396. FR_SFCNTR Field Descriptions

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vsSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	Sync Frames Channel A, even cycle — protocol related variable: size of (*vsSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, odd cycle — protocol related variable: size of (*vsSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, odd cycle — protocol related variable: size of (*vsSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

29.5.2.34 Sync Frame Table Offset Register (FR_SFTOR)

Base + 0x0042 Write: *POC:config*

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

Figure 443. Sync Frame Table Offset Register (FR_SFTOR)

This register defines the flexray memory area related offset for sync frame tables. For more details, see *Section 29.6.12: Sync Frame ID and Sync Frame Deviation Tables*.

Table 397. FR_SFTOR Field Description

Field	Description
SFT_OFFSET	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the flexray memory area. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

29.5.2.35 Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

Base + 0x0044 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM						ELKS	OLKS	EVAL	OVAL	0	0	SDV EN	SID EN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 444. Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see *Section 29.6.12: Sync Frame ID and Sync Frame Deviation Tables*.

Table 398. FR_SFTCCSR Field Descriptions

Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the odd cycle tables. 0 No effect 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).

Table 398. FR_SFTCCSR Field Descriptions (continued)

Field	Description
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OPT	**One Pair Trigger** — This trigger bit controls whether the CC writes continuously or only one pair of Sync Frame Tables into the flexray memory area. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the CC writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the flexray memory area. In this case, the CC clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the CC writes continuously the enabled Sync Frame Tables into the flexray memory area. 0 Write continuously pairs of enabled Sync Frame Tables into flexray memory area. 1 Write only one pair of enabled Sync Frame Tables into flexray memory area.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the CC to write the Sync Frame Deviation Tables into the flexray memory area. 0 Do not write Sync Frame Deviation Tables 1 Write Sync Frame Deviation Tables into flexray memory area **Note:** If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the CC to write the Sync Frame ID Tables into the flexray memory area. 0 Do not write Sync Frame ID Tables 1 Write Sync Frame ID Tables into flexray memory area

29.5.2.36 Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

Base + 0x0046 16-bit write access required Write: Normal Mode

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

Figure 445. Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the CC accepts the sync frame in the current cycle.

Table 399. FR_SFIDRFR Field Descriptions

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see *Section 29.6.15.2: Sync Frame Rejection Filtering*.

29.5.2.37 Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

Base + 0x0048 Write: *POC:config*

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

Figure 446. Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see *Section 29.6.15: Sync Frame Filtering*.

Table 400. FR_SFIDAFVR Field Descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

29.5.2.38 Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

Base + 0x004A Write: *POC:config*

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

Figure 447. Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see *Section 29.6.15.1: Sync Frame Acceptance Filtering*.

Table 401. FR_SFIDAFMR Field Descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

29.5.2.39 Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Base + 0x004C (FR_NMVR0)
Base + 0x004E (FR_NMVR1)
Base + 0x0050 (FR_NMVR2)
Base + 0x0052 (FR_NMVR3)
Base + 0x0054 (FR_NMVR4)
Base + 0x0056 (FR_NMVR5)

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

Figure 448. Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the *Network Management Vector Length Register (FR_NMVLR)*. If FR_NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the *Network Management Vector Registers (FR_NMVR0–FR_NMVR5)*, which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 402. NMVR[0:5] Field Descriptions

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the *Network Management Vector Registers (FR_NMVR0–FR_NMVR5)* and the receive message buffer payload bytes in NMV[0:11] is depicted in *Table 403*.

Table 403. Mapping of NMVRn to the Received Payload Bytes NMVn

NMVRn Register	NMVn Received Payload
FR_NMVR0[NMVP[15:8]]	NMV0
FR_NMVR0[NMVP[7:0]]	NMV1
FR_NMVR1[NMVP[15:8]]	NMV2
FR_NMVR1[NMVP[7:0]]	NMV3
...	
FR_NMVR5[NMVP[15:8]]	NMV10
FR_NMVR5[NMVP[7:0]]	NMV11

29.5.2.40 Network Management Vector Length Register (FR_NMVLR)

Base + 0x0058 Write: *POC:config*

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

Figure 449. Network Management Vector Length Register (FR_NMVLR)

This register defines the length of the network management vector in bytes.

Table 404. FR_NMVLR Field Descriptions

Field	Description
NMVL	**Network Management Vector Length** — protocol related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

29.5.2.41 Timer Configuration and Control Register (FR_TICCR)

Base + 0x005A Write: T2_CFG: *POC:config*
T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_ CFG	T2_ REP	0	0	0	T2ST	0	0	0	T1_ REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 450. Timer Configuration and Control Register (FR_TICCR)

This register is used to configure and control the two timers T1 and T2. For timer details, see *Section 29.6.17: Timer Support*. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 405. FR_TICCR Field Descriptions

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the timebase mode of Timer T2. 0 T2 is absolute timer. 1 T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 T2 is non repetitive 1 T2 is repetitive
T2SP	**Timer T2 Stop** — This trigger bit is used to stop timer T2. 0 no effect 1 stop timer T2
T2TR	**Timer T2 Trigger** — This trigger bit is used to start timer T2. 0 no effect 1 start timer T2

Table 405. FR_TICCR Field Descriptions

Field	Description
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 timer T2 is idle 1 timer T2 is running
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 T1 is non repetitive 1 T1 is repetitive
T1SP	**Timer T1 Stop** — This trigger bit is used to stop timer T1. 0 no effect 1 stop timer T1
T1TR	**Timer T1 Trigger** — This trigger bit is used to start timer T1. 0 no effect 1 start timer T1
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 timer T1 is idle 1 timer T1 is running

Note: Both timers are deactivated immediately when the protocol enters a state different from POC:normal active or POC:normal passive.

29.5.2.42 Timer 1 Cycle Set Register (FR_TI1CYSR)

Base + 0x005C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 451. Timer 1 Cycle Set Register (FR_TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to *Section 29.6.17.1: Absolute Timer T1*.

Table 406. FR_TI1CYSR Field Descriptions

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

29.5.2.43 Timer 1 Macrotick Offset Register (FR_TI1MTOR)

Base + 0x005E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 452. Timer 1 Macrotick Offset Register (FR_TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to *Section 29.6.17.1: Absolute Timer T1*.

Table 407. FR_TI1MTOR Field Descriptions

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

29.5.2.44 Timer 2 Configuration Register 0 (FR_TI2CR0)

Base + 0x0060 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R*		T2_CYC_VAL						R*		T2_CYC_MSK					
W																
R	T2_MTCNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 453. Timer 2 Configuration Register 0 (FR_TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the *Timer Configuration and Control Register (FR_TICCR)*. For a detailed description of timer T2, refer to *Section 29.6.17.2: Absolute / Relative Timer T2*.

Table 408. FR_TI2CR0 Field Descriptions

Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

Note: If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

29.5.2.45 Timer 2 Configuration Register 1 (FR_TI2CR1)

Base + 0x0062 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R*				T2_MTOFFSET											
W																
R	T2_MTCNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 454. Timer 2 Configuration Register 1 (FR_TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the *Timer Configuration and Control Register (FR_TICCR)*. For a detailed description of timer T2, refer to *Section 29.6.17.2: Absolute / Relative Timer T2*.

Table 409. FR_TI2CR1 Field Descriptions

Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

Note: If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

29.5.2.46 Slot Status Selection Register (FR_SSSR)

Base + 0x0064 16-bit write access required Write: Anytime

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

Figure 455. Slot Status Selection Register (FR_SSSR)

This register is used to access the four internal non-memory mapped slot status selection registers FR_SSSR0 to FR_SSSR3. Each internal register selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding *Slot Status Registers (FR_SSR0–FR_SSR7)* according to *Table 411*. For a detailed description of slot status monitoring, refer to *Section 29.6.18: Slot Status Monitoring*.

Table 410. FR_SSSR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 select FR_SSSR0. 01 select FR_SSSR1. 10 select FR_SSSR2. 11 select FR_SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. **Note:** If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 411. Mapping Between FR_SSSRn and FR_SSRn

Internal Slot Status Selection Register	Write the Slot Status of the Slot Selected by FR_SSSRn for each			
	Even Communication Cycle		Odd Communication Cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
FR_SSSR0	FR_SSR0[15:8]	FR_SSR0[7:0]	FR_SSR1[15:8]	FR_SSR1[7:0]
FR_SSSR1	FR_SSR2[15:8]	FR_SSR2[7:0]	FR_SSR3[15:8]	FR_SSR3[7:0]
FR_SSSR2	FR_SSR4[15:8]	FR_SSR4[7:0]	FR_SSR5[15:8]	FR_SSR5[7:0]
FR_SSSR3	FR_SSR6[15:8]	FR_SSR6[7:0]	FR_SSR7[15:8]	FR_SSR7[7:0]

29.5.2.47 Slot Status Counter Condition Register (FR_SSCCR)

Base + 0x0066 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 456. Slot Status Counter Condition Register (FR_SSCCR)

This register is used to access and program the four internal non-memory mapped Slot Status Counter Condition Registers FR_SSCCR0 to FR_SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding *Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)*. The correspondence is given in *Table 413*. For a detailed description of slot status counters, refer to *Section 29.6.18.4: Slot Status Counter Registers*.

Table 412. FR_SSCCR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 select FR_SSCCR0. 01 select FR_SSCCR1. 10 select FR_SSCCR2. 11 select FR_SSCCR3.
CNTCFG	**Counter Configuration** — These bit field controls the channel related incrementing of the slot status counter. 00 increment by 1 if condition is fulfilled on channel A. 01 increment by 1 if condition is fulfilled on channel B. 10 increment by 1 if condition is fulfilled on at least one channel. 11 increment by 2 if condition is fulfilled on both channels channel. increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	**Valid Frame Restriction** — This bit is used to restrict the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit is used to restrict the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit is used to restrict the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	**Startup Frame Restriction** — This bit is used to restrict the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUS MASK[3:0]	**Slot Status Mask** — This bit field is used to enable the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 413. Mapping between internal FR_SSCCRn and FR_SSCRn

Condition Register	Condition Defined for Register
FR_SSCCR0	FR_SSCR0
FR_SSCCR1	FR_SSCR1

Table 413. Mapping between internal FR_SSCCRn and FR_SSCRn

Condition Register	Condition Defined for Register
FR_SSCCR2	FR_SSCR2
FR_SSCCR3	FR_SSCR3

29.5.2.48 Slot Status Registers (FR_SSR0–FR_SSR7)

Base + 0x0068 (FR_SSR0)
Base + 0x006A (FR_SSR1)
Base + 0x006C (FR_SSR2)
Base + 0x006E (FR_SSR3)
Base + 0x0070 (FR_SSR4)
Base + 0x0072 (FR_SSR5)
Base + 0x0074 (FR_SSR6)
Base + 0x0076 (FR_SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 457. Slot Status Registers (FR_SSR0–FR_SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the *Slot Status Selection Register (FR_SSSR)*. Each register is updated after the end of the corresponding slot as shown in *Figure 564*. The register bits are directly related to the protocol variables and described in more detail in *Section 29.6.18: Slot Status Monitoring*.

Table 414. FR_SSR0–FR_SSR7 Field Descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1

Table 414. FR_SSR0–FR_SSR7 Field Descriptions (continued)

Field	Description
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

29.5.2.49 Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Base + 0x0078 (FR_SSCR0)
Base + 0x007A (FR_SSCR1)
Base + 0x007C (FR_SSCR2)
Base + 0x007E (FR_SSCR3)

Additional Reset: RUN Command

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

Figure 458. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register

FR_SSCCRn, which can be programmed by using the *Slot Status Counter Condition Register (FR_SSCCR)*. For more details, see *Section 29.6.18.4: Slot Status Counter Registers*.

Note: If the counter has reached its maximum value 0xFFFF and is in the multicycle mode, i.e. FR_SSCCRn[MCY] = 1, the counter is not reset to 0x0000. The application can reset the counter by clearing the FR_SSCCRn[MCY] bit and waiting for the next cycle start, when the CC clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 415. FR_SSCR0–FR_SSCR3 Field Descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

29.5.2.50 MTS A Configuration Register (FR_MTSACFR)

Base + 0x0080 Write: MTE: Anytime CYCCNTMSK,CYCCNTVAL:*POC:config*

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

Figure 459. MTS A Configuration Register (FR_MTSACFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see *Section 29.6.13: MTS Generation*.

Table 416. FR_MTSACFR Field Descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

29.5.2.51 MTS B Configuration Register (MTSBCFR)

Base + 0x0082 Write: MTE: Anytime CYCCNTMSK,CYCCNTVAL:*POC:config*

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

Figure 460. MTS B Configuration Register (MTSBCFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see *Section 29.6.13: MTS Generation*.

Table 417. MTSBCFR Field Descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

29.5.2.52 Receive Shadow Buffer Index Register (FR_RSBIR)

Base + 0x0084 16-bit write access required Write: WMD, SEL: Any Time
RSBIDX: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	0	0	0	$RSBIDX_{A1}$/$RSBIDX_{A2}$							
W	WMD								$RSBIDX_{B1}$/$RSBIDX_{B2}$							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 461. Receive Shadow Buffer Index Register (FR_RSBIR)

This register is used to provide and retrieve the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to *Section 29.6.6.3.5: Receive Shadow Buffers Concept*.

Table 418. FR_RSBIR Field Descriptions

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	**Selector** — This field is used to select the internal receive shadow buffer index register for access. 00 FR_RSBIR_A1 — receive shadow buffer index register for channel A, segment 1 01 FR_RSBIR_A2 — receive shadow buffer index register for channel A, segment 2 10 FR_RSBIR_B1 — receive shadow buffer index register for channel B, segment 1 11 FR_RSBIR_B2 — receive shadow buffer index register for channel B, segment 2
$RSBIDX_{A1}$ $RSBIDX_{A2}$ $RSBIDX_{B1}$ $RSBIDX_{B2}$	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The CC uses this index to determine the physical location of the shadow buffer header field in the flexray memory area. The CC will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. CC: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index. Legal Values are 0 <= i <= 131. Illegal values will be detected during the message buffer search.

29.5.2.53 Receive FIFO Start Data Offset Register (FR_RFSDOR)

Base + 0x00E6 Write: *POC:config*

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

Figure 462. Receive FIFO Start Data Offset Register (FR_RFSDOR)

Table 419. FR_RFSDOR Field Descriptions

Field	Description
SDO_A SDO_B	**Start Data Field Offset** — This field defines the data field offset of the header field of the first message buffer of the selected FIFO. The CC uses the value of the SDO field to determine the physical location of the receiver FIFO's first message buffer header field. For configuration constraints see *Section 29.7.1.2: Configure Data Field Offsets*.

Note: Since all data fields of the FIFO are of equal length and are located at subsequent system memory addresses the content of the FR_RFSDOR register corresponds to the start address of payload area of the selected FIFO.

29.5.2.54 Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)

Base + 0x00E8 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SMBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 463. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)

Base + 0x00EA Write: Disabled Mode

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

Figure 464. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)

These registers define the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. The system memory base address is used by the BMIF to calculate the physical memory address for system memory accesses for the FIFOs.

Table 420. FR_RFSYMBADR Field Descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defines as a byte address.

29.5.2.55 Receive FIFO Periodic Timer Register (FR_RFPTR)

Base + 0x00EC Write: *POC:config*

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

Figure 465. Receive FIFO Periodic Timer Register (FR_RFPTR)

This register holds periodic timer duration for the periodic FIFO timer. The periodic timer applies to both FIFOs (see *Section 29.6.9.3: FIFO Periodic Timer*).

Table 421. FR_RFPTR Field Descriptions

Field	Description
PTD	**Periodic Timer Duration** — This value defines the periodic timer duration in terms of macroticks. 0000 timer stays expired 3FFF timer never expires other timer expires after specified number of macroticks, expires and is restarted at each cycle start

29.5.2.56 Receive FIFO Watermark and Selection Register (FR_RFWMSR)

Base + 0x0086 Write: *WM_A/WM_B: POC:config, SEL: Anytime*

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

Figure 466. Receive FIFO Watermark and Selection Register (FR_RFWMSR)

This register is used to

- select a receiver FIFO for subsequent programming access through the receiver FIFO configuration registers summarized in *Table 422*.
- to define the watermark for the selected FIFO.

Table 422. SEL Controlled Receiver FIFO Registers

Register
Receive FIFO Start Index Register (FR_RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)
Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Table 423. FR_RFWMSR Field Descriptions

Field	Description
WM_A WM_B	**Watermark** — This field defines the watermark value for the selected FIFO. This value is used to control the generation of the almost full interrupt flags.
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected 1 Receiver FIFO for channel B selected

29.5.2.57 Receive FIFO Start Index Register (FR_RFSIR)

Base + 0x0088 Write: *POC:config*

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

Figure 467. Receive FIFO Start Index Register (FR_RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 424. FR_RFSIR Field Descriptions

Field	Description
$SIDX_A$ $SIDX_B$	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected FIFO. The CC uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

29.5.2.58 Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	$FIFO_DEPTH_A/FIFO_DEPTH_B$								0	$ENTRY_SIZE_A/ENTRY_SIZE_B$						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 468. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, i.e. the number of entries and the size of each entry.

Table 425. RFDSR Field Descriptions

Field	Description
$FIFO_DEPTH_A$ $FIFO_DEPTH_B$	**FIFO Depth** — This field defines the depth of the selected FIFO, i.e. the number of entries. **Note**: If the FIFO_DEPTH is configured to 0, FR_RFFIDRFMR[FIDRFMSK] must be configured to 0 too, to ensure that no frames are received into the FIFO.
$ENTRY_SIZE_A$ $ENTRY_SIZE_B$	**Entry Size** — This field defines the size of the frame data sections for the selected FIFO in 2 byte entities.

29.5.2.59 Receive FIFO A Read Index Register (FR_RFARIR)

Base + 0x008C

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

Figure 469. Receive FIFO A Read Index Register (FR_RFARIR)

This register provides the message buffer header index of the next available FIFO A entry that the application can read.

Table 426. FR_RFARIR Field Descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

Note: If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

29.5.2.60 Receive FIFO B Read Index Register (FR_RFBRIR)

Base + 0x008E

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

Figure 470. Receive FIFO B Read Index Register (FR_RFBRIR)

This register provides the message buffer header index of the next available FIFO B entry that the application can read.

Table 427. FR_RFBRIR Field Descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

Note: If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

29.5.2.61 Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

Base + 0x00EE

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

Figure 471. Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

This register provides the current fill level of the two receiver FIFOs and is used to pop a number of entries from the FIFOs.

Table 428. FR_RFFLPCR Field Descriptions

Field	Description
FLB	**Fill Level FIFO B** — This field provides the current number of entries in the FIFO B.
FLA	**Fill Level FIFO A**— This field provides the current number of entries in the FIFO A.
PCB	**Pop Count FIFO B** — This field defines the number of entries to be removed from FIFO B.
PCA	**Pop Count FIFO A**— This field defines the number of entries to be removed from FIFO A.

Note: If the pop count value PCA/PCB is greater than the current FIFO fill level FLB/FLA, than the FIFO is empty after the update. No notification is given that not the required number of entries was removed.

29.5.2.62 Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

Base + 0x0090 Write: *POC:config*

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

Figure 472. Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

This register defines the filter value for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see *Section 29.6.9.9: FIFO Filtering*.

Table 429. FR_RFMIDAFVR Field Descriptions

Field	Description
$MIDAFVAL_A$ $MIDAFVAL_B$	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

29.5.2.63 Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

Base + 0x0092 Write: *POC:config*

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

Figure 473. Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

This register defines the filter mask for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see *Section 29.6.9.9: FIFO Filtering*.

Table 430. FR_RFMIDAFMR Field Descriptions

Field	Description
$MIDAFMSK_A$ $MIDAFMSK_B$	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

29.5.2.64 Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

Base + 0x0094 Write: *POC:config*

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

Figure 474. Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

This register defines the filter value for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see *Section 29.6.9.9: FIFO Filtering*.

Table 431. FR_RFFIDRFVR Field Descriptions

Field	Description
$FIDRFVAL_A$ $FIDRFVAL_B$	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

29.5.2.65 Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

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

Figure 475. Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see *Section 29.6.9.9: FIFO Filtering*.

Table 432. FR_RFFIDRFMR Field Descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

29.5.2.66 Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

Base + 0x0098 16-bit write access required Write: WMD, IBD, SEL: Any Time
SID: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IBD	SEL		0	SID_A/SID_B										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 476. Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected FIFO. For details on frame ID range filter see *Section 29.6.9.9: FIFO Filtering*.

Table 433. FR_RFRFCFR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 program lower interval boundary 1 program upper interval boundary
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 select frame ID range filter 0. 01 select frame ID range filter 1. 10 select frame ID range filter 2. 11 select frame ID range filter 3.
SID_A SID_B	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

29.5.2.67 Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Base + 0x009A Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 477. Receive FIFO Range Filter Control Register (FR_RFRFCTR)

This register is used to enable and disable each frame ID range filter and to define whether it is running as acceptance or rejection filter.

Table 434. FR_RFRFCTR Field Descriptions

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 range filter 3 runs as acceptance filter 1 range filter 3 runs as rejection filter
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 range filter 2 runs as acceptance filter 1 range filter 2 runs as rejection filter
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 range filter 1 runs as acceptance filter 1 range filter 1 runs as rejection filter
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 range filter 0 runs as acceptance filter 1 range filter 0 runs as rejection filter
F3EN	**Range Filter 3 Enable** — This control bit is used to enable and disable the frame ID range filter 3. 0 range filter 3 disabled 1 range filter 3 enabled
F2EN	**Range Filter 2 Enable** — This control bit is used to enable and disable the frame ID range filter 2. 0 range filter 2 disabled 1 range filter 2 enabled
F1EN	**Range Filter 1 Enable** — This control bit is used to enable and disable the frame ID range filter 1. 0 range filter 1 disabled 1 range filter 1 enabled
F0EN	**Range Filter 0 Enable** — This control bit is used to enable and disable the frame ID range filter 0. 0 range filter 0 disabled 1 range filter 0 enabled

29.5.2.68 Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

Base + 0x009C

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

Figure 478. Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 435. FR_LDTXSLAR Field Descriptions

Field	Description
LASTDYNTX SLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

29.5.2.69 Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

Base + 0x009E

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

Figure 479. Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 436. FR_LDTXSLBR Field Descriptions

Field	Description
LASTDYNT X SLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

29.5.2.70 Protocol Configuration Registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in *Table 437*. For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Table 437. Protocol Configuration Register Fields

Name	Description[1]	Min	Max	Unit	FR_PCR
coldstart_attempts	gColdstartAttempts			number	3
action_point_offset	*gdActionPointOffset* - 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* - 1			gdBit	4
dynamic_slot_idle_phase	gdDynamicSlotIdlePhase			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* - 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* - 1			MT	2
static_slot_length	gdStaticSlot			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* - 1			MT	13

Table 437. Protocol Configuration Register Fields (continued)

Name	Description(1)	Min	Max	Unit	FR_PCR
symbol_window_exists	*gdSymbolWindow*!=0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	gdTSSTransmitter			gdBit	5
wakeup_symbol_rx_idle	gdWakeupSymbolRxIdle			gdBit	5
wakeup_symbol_rx_low	gdWakeupSymbolRxLow			gdBit	3
wakeup_symbol_rx_window	gdWakeupSymbolRxWindow			gdBit	4
wakeup_symbol_tx_idle	gdWakeupSymbolTxIdle			gdBit	8
wakeup_symbol_tx_low	gdWakeupSymbolTxLow			gdBit	5
noise_listen_timeout	(*gListenNoise* * *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	pMacroInitialOffset[A]			MT	6
macro_initial_offset_b	pMacroInitialOffset[B]			MT	16
macro_per_cycle	gMacroPerCycle			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28
max_without_clock_correction_fatal	gMaxWithoutClockCorrectionFatal			cyclepairs	8
max_without_clock_correction_passive	gMaxWithoutClockCorrectionPassive			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	gNumberOfStaticSlots			static slot	2
offset_correction_start	gOffsetCorrectionStart			MT	11
payload_length_static	gPayloadLengthStatic			2-bytes	19
max_payload_length_dynamic	pPayloadLengthDynMax			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	pAllowHaltDueToClock			bool	26
allow_passive_to_active	pAllowPassiveToActive			cyclepairs	12
cluster_drift_damping	pClusterDriftDamping			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	pKeySlotId			number	18
key_slot_used_for_startup	pKeySlotUsedForStartup			bool	11
key_slot_used_for_sync	pKeySlotUsedForSync			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21

Table 437. Protocol Configuration Register Fields (continued)

Name	Description(1)	Min	Max	Unit	FR_PCR
sync_node_max	gSyncNodeMax			number	30
micro_initial_offset_a	pMicroInitialOffset[A]			µT	20
micro_initial_offset_b	pMicroInitialOffset[B]			µT	20
micro_per_cycle	pMicroPerCycle			µT	22/23
micro_per_cycle_min	pMicroPerCycle - pdMaxDrift			µT	24/25
micro_per_cycle_max	pMicroPerCycle + pdMaxDrift			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	pOffsetCorrectionOut			µT	9
rate_correction_out	pRateCorrectionOut			µT	14
single_slot_enabled	pSingleSlotEnabled			bool	10
wakeup_channel	pWakeupChannel	see *Table 438*			10
wakeup_pattern	pWakeupPattern			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	pExternOffsetCorrection			µT	29
extern_rate_correction	pExternRateCorrection			µT	21

1. See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 438. Wakeup Channel Selection

wakeup_channel	Wakeup Channel
0	A
1	B

29.5.2.70.1 Protocol Configuration Register 0 (FR_PCR0)

Base + 0x00A0 Write: *POC:config*

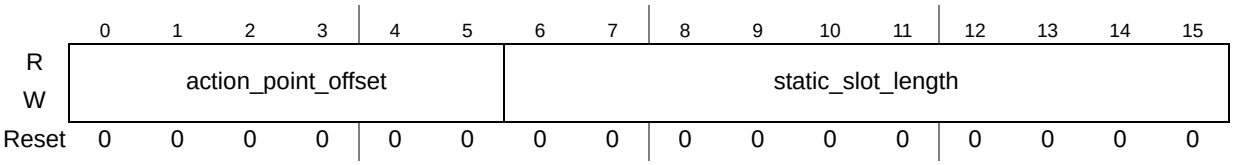


Figure 480. Protocol Configuration Register 0 (FR_PCR0)

29.5.2.70.2 Protocol Configuration Register 1 (FR_PCR1)

Base + 0x00A2 Write: *POC:config*

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

Figure 481. Protocol Configuration Register 1 (FR_PCR1)

29.5.2.70.3 Protocol Configuration Register 2 (FR_PCR2)

Base + 0x00A4 Write: *POC:config*

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

Figure 482. Protocol Configuration Register 2 (FR_PCR2)

29.5.2.70.4 Protocol Configuration Register 3 (FR_PCR3)

Base + 0x00A6 Write: *POC:config*

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

Figure 483. Protocol Configuration Register 3 (FR_PCR3)

29.5.2.70.5 Protocol Configuration Register 4 (FR_PCR4)

Base + 0x00A8 Write: *POC:config*

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

Figure 484. Protocol Configuration Register 4 (FR_PCR4)

29.5.2.70.6 Protocol Configuration Register 5 (FR_PCR5)

Base + 0x00AA Write: *POC:config*

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

Figure 485. Protocol Configuration Register 5 (FR_PCR5)

29.5.2.70.7 Protocol Configuration Register 6 (FR_PCR6)

Base + 0x00AC Write: *POC:config*

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

Figure 486. Protocol Configuration Register 6 (FR_PCR6)

29.5.2.70.8 Protocol Configuration Register 7 (FR_PCR7)

Base + 0x00AE Write: *POC:config*

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

Figure 487. Protocol Configuration Register 7 (FR_PCR7)

29.5.2.70.9 Protocol Configuration Register 8 (FR_PCR8)

Base + 0x00B0 Write: *POC:config*

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

Figure 488. Protocol Configuration Register 8 (FR_PCR8)

29.5.2.70.10 Protocol Configuration Register 9 (FR_PCR9)

Base + 0x00B2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	mini slot_ exists	sym bol_ win dow_ exists	offset_correction_out													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 489. Protocol Configuration Register 9 (FR_PCR9)

29.5.2.70.11 Protocol Configuration Register 10 (FR_PCR10)

Base + 0x00B4 Write: *POC:config*

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

Figure 490. Protocol Configuration Register 10 (FR_PCR10)

29.5.2.70.12 Protocol Configuration Register 11 (FR_PCR11)

Base + 0x00B6 Write: *POC:config*

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

Figure 491. Protocol Configuration Register 11 (FR_PCR11)

29.5.2.70.13 Protocol Configuration Register 12 (FR_PCR12)

Base + 0x00B8 Write: *POC:config*

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

Figure 492. Protocol Configuration Register 12 (FR_PCR12)

29.5.2.70.14 Protocol Configuration Register 13 (FR_PCR13)

Base + 0x00BA Write: *POC:config*

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

Figure 493. Protocol Configuration Register 13 (FR_PCR13)

29.5.2.70.15 Protocol Configuration Register 14 (FR_PCR14)

Base + 0x00BC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	rate_correction_out												listen_timeout[20:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 494. Protocol Configuration Register 14 (FR_PCR14)

29.5.2.70.16 Protocol Configuration Register 15 (FR_PCR15)

Base + 0x00BE Write: *POC:config*

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

Figure 495. Protocol Configuration Register 15 (FR_PCR15)

29.5.2.70.17 Protocol Configuration Register 16 (FR_PCR16)

Base + 0x00C0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	macro_initial_offset_b								noise_listen_timeout[24:16]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 496. Protocol Configuration Register 16 (FR_PCR16)

29.5.2.70.18 Protocol Configuration Register 17 (FR_PCR17)

Base + 0x00C2 Write: *POC:config*

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

Figure 497. Protocol Configuration Register 17 (FR_PCR17)

29.5.2.70.19 Protocol Configuration Register 18 (FR_PCR18)

Base + 0x00C4 Write: *POC:config*

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

Figure 498. Protocol Configuration Register 18 (FR_PCR18)

29.5.2.70.20 Protocol Configuration Register 19 (FR_PCR19)

Base + 0x00C6 Write: *POC:config*

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

Figure 499. Protocol Configuration Register 19 (FR_PCR19)

29.5.2.70.21 Protocol Configuration Register 20 (FR_PCR20)

Base + 0x00C8 Write: *POC:config*

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

Figure 500. Protocol Configuration Register 20 (FR_PCR20)

29.5.2.70.22 Protocol Configuration Register 21 (FR_PCR21)

Base + 0x00CA Write: *POC:config*

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

Figure 501. Protocol Configuration Register 21 (FR_PCR21)

29.5.2.70.23 Protocol Configuration Register 22 (FR_PCR22)

Base + 0x00CC Write: *POC:config*

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

Figure 502. Protocol Configuration Register 22 (FR_PCR22)

29.5.2.70.24 Protocol Configuration Register 23 (FR_PCR23)

Base + 0x00CE Write: *POC:config*

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

Figure 503. Protocol Configuration Register 23 (FR_PCR23)

29.5.2.70.25 Protocol Configuration Register 24 (FR_PCR24)

Base + 0x00D0 Write: *POC:config*

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

Figure 504. Protocol Configuration Register 24 (FR_PCR24)

29.5.2.70.26 Protocol Configuration Register 25 (FR_PCR25)

Base + 0x00D2 Write: *POC:config*

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

Figure 505. Protocol Configuration Register 25 (FR_PCR25)

29.5.2.70.27 Protocol Configuration Register 26 (FR_PCR26)

Base + 0x00D4 Write: *POC:config*

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

Figure 506. Protocol Configuration Register 26 (FR_PCR26)

29.5.2.70.28 Protocol Configuration Register 27 (FR_PCR27)

Base + 0x00D6 Write: *POC:config*

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

Figure 507. Protocol Configuration Register 27 (FR_PCR27)

29.5.2.70.29 Protocol Configuration Register 28 (FR_PCR28)

Base + 0x00D8 Write: *POC:config*

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

Figure 508. Protocol Configuration Register 28 (FR_PCR28)

29.5.2.70.30 Protocol Configuration Register 29 (FR_PCR29)

Base + 0x00DA Write: *POC:config*

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

Figure 509. Protocol Configuration Register 29 (FR_PCR29)

29.5.2.70.31 Protocol Configuration Register 30 (FR_PCR30)

Base + 0x00DC Write: *POC:config*

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

Figure 510. Protocol Configuration Register 30 (FR_PCR30)

29.5.2.71 ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

Base + 0x00F0 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRNE_OF	LRCE_OF	DRNE_OF	DRCE_OF	LRNE_IF	LRCE_IF	DRNE_IF	DRCE_IF	0	0	0	0	LRNE_IE	LRCE_IE	DRNE_IE	DRCE_IE
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

Figure 511. ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

This register provides the means to control the ECC related interrupt request lines and provides the corresponding interrupt flags. The interrupt flags are cleared by writing 1, which resets the corresponding report registers. For a detailed description see *Section 29.6.24.2: Memory Error Reporting*.

Table 439. FR_EEIFER Field Descriptions

Field	Description
Error Overflow Flags	
LRNE_OF	**LRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *but not corrected* on CHI LRAM and interrupt flag LRNE_IF is already 1. b) memory errors are detected *but not corrected* on at least two banks of CHI LRAM 0 no such event 1 Non-Corrected Error overflow detected on CHI LRAM
LRCE_OF	**LRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *and corrected* on CHI LRAM and interrupt flag LRCE_IF is already 1. b) memory errors are detected *and corrected* on at least two banks of CHI LRAM 0 no such event 1 Corrected Error overflow detected on CHI LRAM Note: Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_OF	**DRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *but not corrected* on PE DRAM and interrupt flag DRNE_IF is already 1. b) memory errors are detected *but not corrected* on at least two banks of the PE DRAM 0 no such event 1 Non-Corrected Error overflow detected on PE DRAM
DRCE_OF	**DRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *and corrected* on PE DRAM and interrupt flag DRCE_IF is already 1. b) memory errors are detected *and corrected* on at least two banks of PE DRAM 0 no such event 1 Corrected Error overflow detected on PE DRAM
Error Interrupt Flags	
LRNE_IF	**LRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected but not corrected* on the CHI LRAM. 0 no such event 1 Non-Corrected Error detected on CHI LRAM
LRCE_IF	**LRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected and corrected* on the CHI LRAM. 0 no such event 1 Corrected Error detected on CHI LRAM Note: Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_IF	**DRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected but not corrected* on PE DRAM. 0 no such event 1 Non-Corrected Error detected on PE DRAM
DRCE_IF	**DRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected and corrected* on PE DRAM. 0 no such event 1 Corrected Error detected on PE DRAM

Table 439. FR_EEIFER Field Descriptions (continued)

Field	Description
Error Interrupt Enables	
LRNE_IE	**LRAM Non-Corrected Error Interrupt Enable** — This flag controls if the LRAM Non-Corrected Error Interrupt line is asserted when the LRNE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
LRCE_IE	**LRAM Corrected Error Interrupt Enable** — This flag controls if the LRAM Corrected Error Interrupt line is asserted when the LRCE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
DRNE_IE	**DRAM Non-Corrected Error Interrupt Enable** — This flag controls if the DRAM Non-Corrected Error Interrupt line is asserted when the DRNE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
DRCE_IE	**DRAM Corrected Error Interrupt Enable** — This flag controls if the DRAM Corrected Error Interrupt line is asserted when the DRCE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line

29.5.2.72 ECC Error Report and Injection Control Register (FR_EERICR)

Base + 0x00F2

Write: ERS: Anytime
ERM, EIM, EIE: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	0	0	0	0	ERS		0	0	0	ERM	0	0	EIM	EIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 512. ECC Error Report and Injection Control Register (FR_EERICR)

This register configures the error injection and error reporting and provides the selector for the content of the report registers.

Table 440. FR_EERICR Field Descriptions

Field	Description
BSY	**Register Update Busy**— This field indicates the current state of the ECC configuration update and controls the register write access condition IDL specified in *Section 29.5.2.2: Register Write Access* 0 ECC configuration is idle 1 ECC configuration is running
ERS	**Error Report Select** — This field selects the content of the ECC Error reporting registers. 00 show PE DRAM non-corrected error information 01 show PE DRAM corrected error information 10 show CHI LRAM non-corrected error information 11 show CHI LRAM corrected error information
ERM	**Error Report Mode** — This bit configures the type of data written into the internal error report registers on the detection of a memory error. 0 store data and code as delivered by ecc decoding logic. 1 store data and code as read from the memory.
EIM	**Error Injection Mode** — This bit configures the ECC error injection mode. 0 use FR_EEIDR[DATA] and FR_EEICR[CODE] as XOR distortion pattern for error injection. 1 use FR_EEIDR[DATA] and FR_EEICR[CODE] as write value for error injection.
EIE	**Error Injection Enable** — This bit configures the ECC error injection on the memories. 0 Error injection disabled 1 Error injection enabled

29.5.2.73 ECC Error Report Address Register (FR_EERAR)

Base + 0x00F4

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

Figure 513. ECC Error Report Address Register (FR_EERAR)

This register provides the memory identifier, bank, and address for which the memory error is reported.

Table 441. FR_EERAR Field Descriptions

Field	Description
MID	**Memory Identifier** — This flag provides the memory instance for which the memory error is reported. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field provides the BANK for which the memory error is reported. 111 reset value, indicates no error found after reset. For MID=0: 000 PE DRAM [7:0] 001 PE DRAM [15:8] others - not used For MID=1: Refer to *Table 442* for the assignment of the LRAM banks.
ADDR	**Memory Address** — This field provides the address of the failing memory location.

Table 442. LRAM Bank Value for MID = 1

BANK	Register		
000	FR_MBCCFR(2n)	FR_MBDOR(6n)	FR_LEETR0
001	FR_MBFIDR(2n)	FR_MBDOR(6n + 1)	FR_LEETR1
010	FR_MBIDXR(2n)	FR_MBDOR(6n + 2)	FR_LEETR2
011	FR_MBCCFR(2n+1)	FR_MBDOR(6n + 3)	FR_LEETR3
100	FR_MBFIDR(2n+1)	FR_MBDOR(6n + 4)	FR_LEETR4
101	FR_MBIDXR(2n+1)	FR_MBDOR(6n + 5)	FR_LEETR5
110	Not Used	Not Used	Not Used
111			

29.5.2.74 ECC Error Report Data Register (FR_EERDR)

Base + 0x00F6

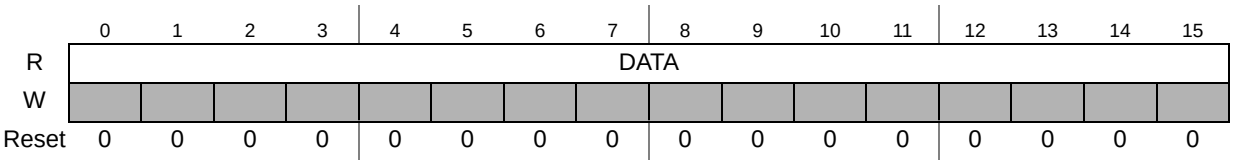


Figure 514. ECC Error Report Data Register (FR_EERDR)

This register provides the data related information of the reported memory read access. The assignment of the bits depends on the selected memory and memory bank as shown in *Table 444*.

Table 443. FR_EERDR Field Descriptions

Field	Description
DATA	**Data** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM=0: Ecc Data, shows data as generated by the ecc decoding logic. ERM=1: Memory Data, shows data as read from the memory.

Table 444. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field

MEM	BANK	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PE DRAM	0									PE DRAM[7:0]							
PE DRAM	1									PE DRAM[15:8]							
CHI LRAM	0	FR_MBCCFR(2n)															
CHI LRAM	0	FR_MBDOR(6n)															
CHI LRAM	0	FR_LEETR0															
CHI LRAM	1						FR_MBFIDR(2n)[FID]										
CHI LRAM	1	FR_MBDOR(6n+1)															
CHI LRAM	1	FR_LEETR1															

Table 444. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field

MEM	BANK	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHI LRAM	2									FR_MBIDXR(2n)[MBIDX]							
CHI LRAM	2	FR_MBDOR(6n+2)															
CHI LRAM	2	FR_LEETR2															
CHI LRAM	3	FR_MBCCFR(2n+1)															
CHI LRAM	3	FR_MBDOR(6n+3)															
CHI LRAM	3	FR_LEETR3															
CHI LRAM	4						FR_MBFIDR(2n+1)[FID]										
CHI LRAM	4	FR_MBDOR(6n+4)															
CHI LRAM	4	FR_LEETR4															
CHI LRAM	5									FR_MBIDXR(2n+1)[MBIDX]							
CHI LRAM	5	FR_MBDOR(6n+5)															
CHI LRAM	5	FR_LEETR5															

29.5.2.75 ECC Error Report Code Register (FR_EERCR)

Base + 0x00F8

	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	CODE				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 515. ECC Error Report Code Register (FR_EERCR)

This register provides the ecc related information of the reported memory read access.

Table 445. FR_EERSR Field Descriptions

Field	Description
CODE	**Code** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM=0: Syndrome. Shows the ecc syndrome generated by the ecc decoding logic. The coding of the PE DRAM syndrome is shown in *Section 29.6.24.2.2: PE DRAM Syndrome* The coding of the CHI LRAM syndrome is shown in *Section 29.6.24.2.4: CHI LRAM Syndrome*. ERM=1: Checkbits. Shows the ecc checkbits read from the memory.

29.5.2.76 ECC Error Injection Address Register (FR_EEIAR)

Base + 0x00FA Write: IDL

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

Figure 516. ECC Error Injection Address Register (FR_EEIAR)

This register defines the memory module, bank, and address where the ECC error has to be injected.

Table 446. FR_EEIAR Field Descriptions

Field	Description
MID	**Memory Identifier** — This flag defines the memory instance for ECC error injection. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field defines the memory bank for ECC error injection. For MID=0: 000 BANK0: PE DRAM [7:0] 001 BANK1: PE DRAM [15:8] others reserved For MID=1: Refer to *Table 442* for the assignment of the LRAM banks.
ADDR	**Memory Address** — This flag defines the memory address for ECC error injection.

29.5.2.77 ECC Error Injection Data Register (FR_EEIDR)

Base + 0x00FC Write: IDL

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

Figure 517. ECC Error Injection Data Register (FR_EEIDR)

This register defines the data distortion pattern for the error injection write. The number of valid bits depends on the selected memory and memory bank as shown in *Table 444*.

Table 447. FR_EEIDR Field Descriptions

Field	Description
DATA	**Data** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM=0: This field defines the XOR distortion pattern for the data written into the memory. EIM=1: This field defines the data to be written into the memory.

Note: The effect of the error injected depends from the LRAM content at the address accessed and from the module internal usage of the data. Refer to Section 29.6.24.3: Memory Error Response for details.

29.5.2.78 ECC Error Injection Code Register (FR_EEICR)

Base + 0x00FE Write: IDL

	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	CODE				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 518. ECC Error Injection Code Register (FR_EEICR)

This register defines the ecc code distortion pattern for the error injection write.

Table 448. FR_EEICR Field Descriptions

Field	Description
CODE	**Code** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM=0: This field defines the XOR distortion pattern for the ecc checkbits written into the memory. EIM=1: This field defines the ecc checkbits written into the memory.

29.5.2.79 Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

Base + 0x0800 (FR_MBCCSR0)
Base + 0x0808 (FR_MBCCSR1)
...
Base + 0x0BF8
(FR_MBCCSR127)

Write: MTD: *POC:cnfig* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 519. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in *Section 29.6.6: Individual Message Buffer Functional Description*

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 449. FR_MBCCSRn Field Descriptions

Field	Description
Message Buffer Configuration	
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer 1 Transmit message buffer
Message Buffer Control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission 1 Message buffer data ready for transmission
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value of the EDS status bit. 0 No effect 1 Message buffer enable or disable is triggered
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit and writes 0 to the EDT bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect 1 Message buffer lock or unlock is triggered
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled 1 Interrupt request generation enabled
Message Buffer Status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame Header and Message buffer data field not updated 1 Frame Header and Message buffer data field updated
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 receive message buffer contains no valid frame data / message is transmitted for the first time 1 receive message buffer contains valid frame data / message will be transferred again
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event 1 Slot status field updated or transmit message buffer just enabled

29.5.2.80 Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

Base + 0x0802 (FR_MBCCFR0)
Base + 0x080A (FR_MBCCFR1)
...
Base + 0x0BFA
(FR_MBCCFR127)

16-bit write access required

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MTM	CHA	CHB	CCFE	CCFMSK								CCFVAL			
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 520. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to *Section 29.6.7.1: Message Buffer Cycle Counter Filtering*.

Table 450. FR_MBCCFRn Field Descriptions

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode 1 State transmission mode
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to *Table 451*.
CCFE	**Cycle Counter Filtering Enable** — This control bit is used to enable and disable the cycle counter filtering. 0 Cycle counter filtering disabled 1 Cycle counter filtering enabled
CCFMSK	Cycle Counter Filtering Mask — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 451. Channel assignment description

CHA	CHB	Transmit Message Buffer		Receive Message Buffer	
		static segment	dynamic segment	static segment	dynamic segment
1	1	transmit on both channel A and channel B	reserved; functionality not guaranteed	store first valid frame received on either channel A or channel B	reserved; functionality not guaranteed
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

Note: If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise,

the message buffer configuration is illegal and the result of the message buffer search is not defined.

29.5.2.81 Message Buffer Frame ID Registers (FR_MBFIDRn)

Base + 0x0804 (FR_MBFIDR0)
Base + 0x080C (FR_MBFIDR1)
...
Base + 0x0BFC
(FR_MBFIDR127)

16-bit write access required

Write: *POC:config* or MB_DIS

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

Figure 521. Message Buffer Frame ID Registers (FR_MBFIDRn)

Table 452. FR_MBFIDRn Field Descriptions

Field	Description
FID	**Frame ID** — The semantic of this field depends on the message buffer transfer type. – *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. – *Transmit Message Buffer:* This field is used to determine the slot in which the message in this message buffer should be transmitted.

29.5.2.82 Message Buffer Index Registers (FR_MBIDXRn)

Base + 0x0806 (FR_MBIDXR0)
Base + 0x080E (FR_MBIDXR1)
...
Base + 0x0BFE
(FR_MBIDXR127)

16-bit write access required

Write: *POC:config* or MB_DIS

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

Figure 522. Message Buffer Index Registers (FR_MBIDXRn)

Table 453. FR_MBIDXRn Field Descriptions

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The CC updates this register after frame reception or transmission. Legal Values are 0 <= i <= 131. Illegal values will be detected during the message buffer search.

29.5.2.83 Message Buffer Data Field Offset Registers (FR_MBDORn)

Base + 0x1000 (FR_MBDOR0) 16-bit write access required Write: *POC:config*
Base + 0x1002 (FR_MBDOR1)
...
Base + 0x1106 (FR_MBDOR131)



Figure 523. Message Buffer Data Field Offset Registers (FR_MBDORn)

Table 454. FR_MBDORn Field Descriptions

Field	Description
MBDO	**Message Buffer Data Field Offset** — This field provides the data field offset belonging to a particular Message Buffer Index. For configuration constraints see *Section 29.7.1.2: Configure Data Field Offsets*.

29.5.2.84 LRAM ECC Error Test Registers (FR_LEETRn)

Base + 0x1108 (FR_LEETR0) 16-bit write access required Write: Anytime
...
Base + 0x1112 (FR_LEETR5)

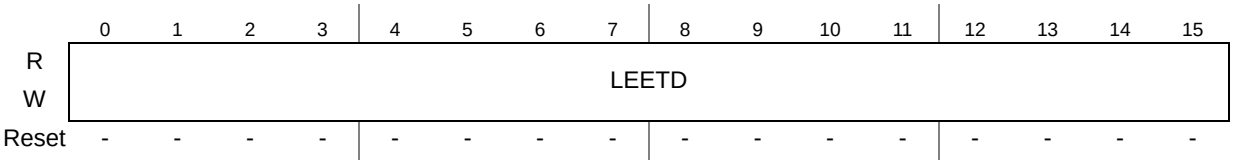


Figure 524. LRAM ECC Error Test Registers (FR_LEETRn)

Table 455. FR_LEETRn Field Descriptions

Field	Description
LEETD	**LRAM ECC Error Test Data** — This field contains the LRAM data belonging to the test register located in LRAM Bank n.

29.6 Functional Description

This section provides a detailed description of the functionality implemented in the CC.

29.6.1 Message Buffer Concept

The CC uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in *Section 29.6.3: Message Buffer Types*. The physical message buffer is located in the flexray memory area and is described in *Section 29.6.2: Physical Message Buffer*.

29.6.2 Physical Message Buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the flexray memory area.The structure of a physical message buffer is depicted in *Figure 525*.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header* and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

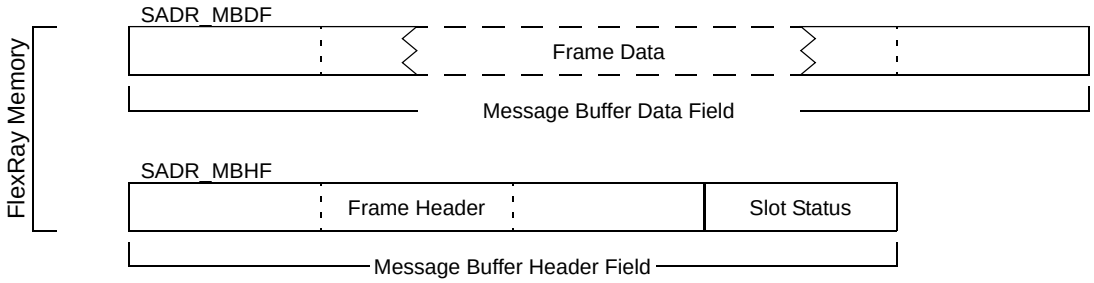


Figure 525. Physical Message Buffer Structure

29.6.2.1 Message Buffer Header Field

The message buffer header field is a contiguous region in the flexray memory area and occupies eight bytes. It contains the frame header, and the slot status. Its structure is shown in *Figure 525*. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

29.6.2.1.1 Frame Header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the usage and the content of the frame header is provided in *Section 29.6.5.2.1: Frame Header Description*.

29.6.2.1.2 Slot Status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in *Section 29.6.5.2.2: Slot Status Description*.

29.6.2.2 Message Buffer Data Field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in *Section 29.6.3: Message Buffer Types*.

29.6.3 Message Buffer Types

The CC provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

29.6.3.1 Individual Message Buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The CC supports three types of individual message buffers, which are described in *Section 29.6.6: Individual Message Buffer Functional Description*.

Each individual message buffer consists of two parts, the physical message buffer, which is located in the flexray memory area, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in *Figure 526*.

Each individual message buffer has a message buffer number n assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number *n* is controlled by the registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the *Message Buffer Index Registers (FR_MBIDXRn)*. The start address SADR_MBHF of the related message buffer header field in the flexray memory area is determined according to *Equation 21*.

Equation 21 SADR_MBHF = (FR_MBIDXRn[MBIDX] * 8) + SMBA

The data field belonging to a particular physical message buffer is characterized by the data field offset. For each physical message buffer with MBIDX i the FR_MBDORi contains the offset of the corresponding message buffer data field with respect to the CC flexray memory area base address as provided by SMBA field in the *System Memory Base Address Register (FR_SYMBADR)*.

The data field offset is used to determine the start address *SADR_MBDF* of the corresponding message buffer data field in the flexray memory area according to *Equation 22*.

Equation 22 SADR_MBDF = [Data Field Offset] + SMBA

The FR_MBDORn are stored in the module internal memory LRAM. Refer to *Section 29.7.2.3: CHI LRAM Initialization* for the setup of the data field offset values.

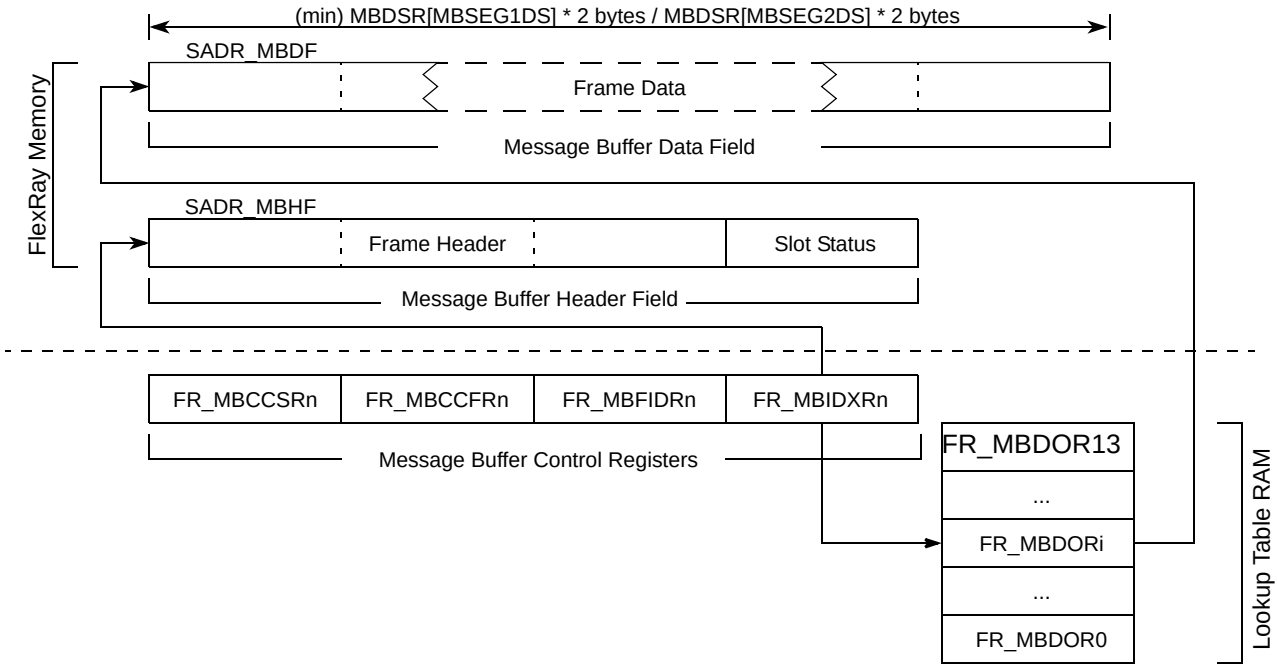


Figure 526. Individual Message Buffer Structure

29.6.3.1.1 Individual Message Buffer Segments

The set of the individual message buffers can be split up into two message buffer segments using the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*. All individual message buffers with a message buffer number n <= FR_MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number n > FR_MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- all physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length
- the minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 * FR_MBDSR[MBSEG1DS] bytes
- the minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 * FR_MBDSR[MBSEG2DS] bytes.

29.6.3.2 Receive Shadow Buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The CC provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the flexray memory area and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in *Figure 527*. The four internal shadow buffer control registers can be accessed by the *Receive Shadow Buffer Index Register (FR_RSBIR)*.

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the *Receive Shadow Buffer Index Register (FR_RSBIR)*. The start address SADR_MBHF of the related message buffer header field in the flexray memory area is determined according to *Equation 23*.

Equation 23SADR_MBHF = (FR_RSBIR[RSBIDX] * 8) + SMBA

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in *Receive Shadow Buffer Index Register (FR_RSBIR)*.

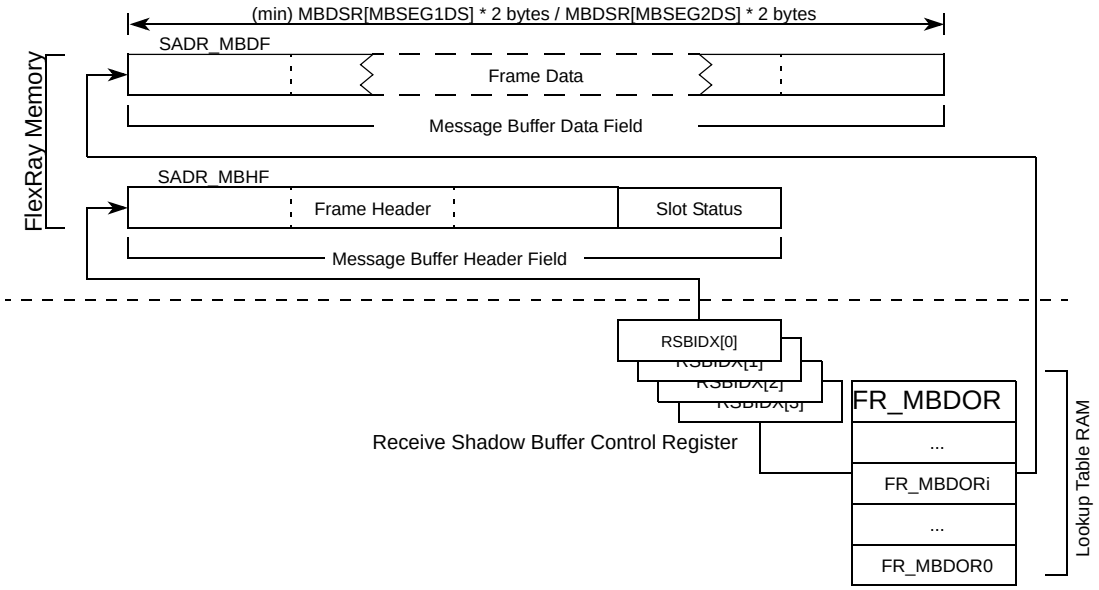


Figure 527. Receive Shadow Buffer Structure

29.6.3.3 Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The CC provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the flexray memory area and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in *Figure 528*.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the *Receive FIFO Start Index Register (FR_RFSIR)*, the *Receive FIFO Depth and Size Register (RFDSR)*, and the *Receive FIFO A Read Index Register (FR_RFARIR)* / *Receive FIFO B Read Index Register (FR_RFBRIR)*.

The system memory base address SMBA valid for the receive FIFOs is defined by the system memory base address register selected by the FIFO address mode bit FR_MCR[FAM], refer to *Section 29.5.2.4: Module Configuration Register (FR_MCR)*.

The start byte address SADR_MBHF[1] of the first message buffer header field that belongs to the receive FIFO is determined according to *Equation 24*.

Equation 24
SADR_MBHF[1] = (8 * FR_RFSIR[SIDX]) + SMBA

The start byte address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the flexray memory area is determined according to *Equation 25*.

Equation 25
SADR_MBHF[n] = (8 * (FR_RFSIR[SIDX] + RFDSR[FIFO_DEPTH])) + SMBA

The required information to access the current entry of the FIFO is given in the following registers:

- The registers *Receive FIFO A Read Index Register (FR_RFARIR)* and *Receive FIFO B Read Index Register (FR_RFBRIR)* provide the index of the physical message buffer belonging to the current entry.

  The data field offset belonging to the current FIFO entry RF_DFO[X] must be calculated using the current read index i according to the following formula:

Equation 26
RF_DFO[X] = FR_RFSDOR[X] + (FR_RFDSR[X][ENTRY_SIZE] * 2) * i - FR_RFSIDX[X})

Note: The current read index loops up starting at the number given in the FR_RD[A/B]RDIDX register for the required number of entries.

Refer to Section 29.6.9.8: FIFO Update for details about updating the FIFO read pointer.

All message buffer header fields assigned to a receive FIFO are within a contiguous region defined by FR_RFSIR[SIDX] and RFDSR[FIFO_DEPTH].

The data sections of all FIFO entries within on receive FIFO are of the same length defined by RFDSR[FIFO_SIZE].

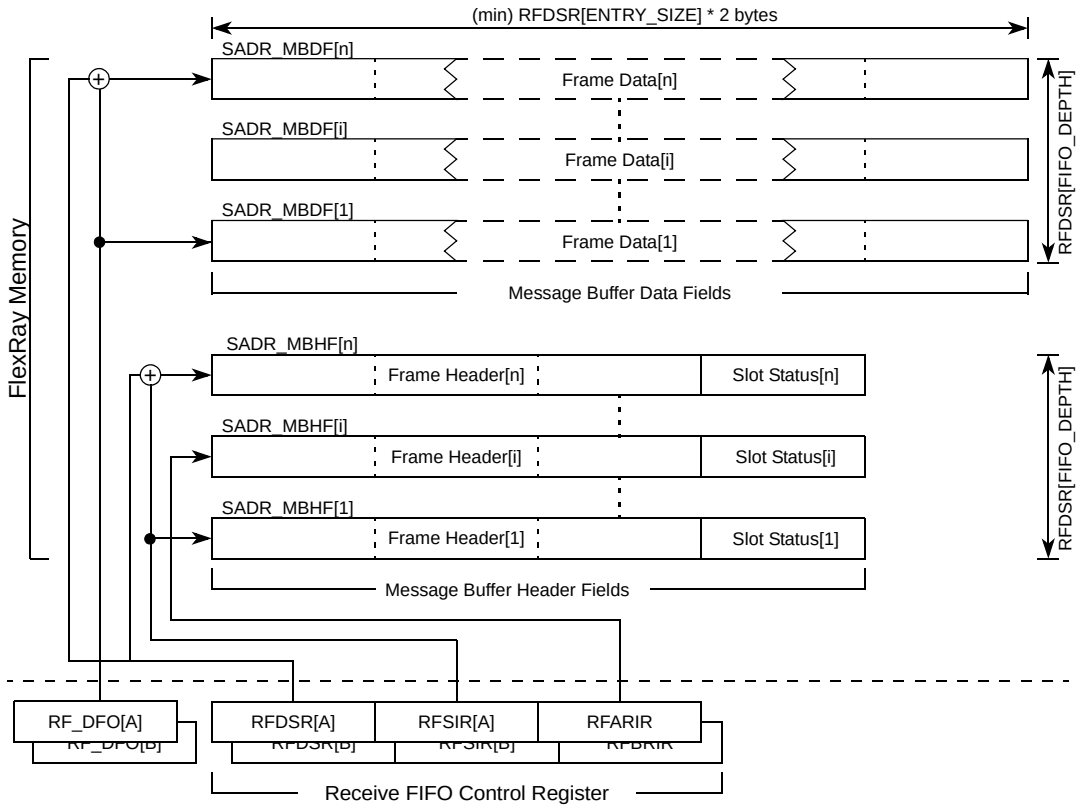


Figure 528. Receive FIFO Structure

Note: The actual values of the data field offsets RF_DFO[A/B] need to be calculated according to Equation 26. They are not stored in a register.

29.6.3.4 Message Buffer Configuration and Control Data

This section describes the configuration and control data for each message buffer type.

29.6.3.4.1 Individual Message Buffer Configuration Data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

29.6.3.4.1.1 Common Configuration Data

The set of common configuration data for individual message buffers is located in the following registers.

- *Message Buffer Data Size Register (FR_MBDSR)*
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see *Section 29.6.3.1.1: Individual Message Buffer Segments*

29.6.3.4.1.2 Specific Configuration Data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*
  The MTD bit configures the message buffer type.
- *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- *Message Buffer Frame ID Registers (FR_MBFIDRn)*
  For a transmit message buffer, the FID field is used to determine the slot in which the message in this message buffer will be transmitted.
- *Message Buffer Index Registers (FR_MBIDXRn)*
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

29.6.3.5 Individual Message Buffer Control Data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*.

29.6.3.6 Receive Shadow Buffer Configuration Data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the *Receive Shadow Buffer Index Register (FR_RSBIR)*. For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full CC control.

29.6.3.7 Receive FIFO Control and Configuration Data

This section describes the configuration and control data for the two receive FIFOs.

29.6.3.7.1 Receive FIFO Configuration Data

The CC provides two functional independent receive FIFOs, one per channel. The FIFOs have a common subset of configuration data:

- *Receive FIFO Periodic Timer Register (FR_RFPTR)*

Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- *Receive FIFO Watermark and Selection Register (FR_RFWMSR)*
- *Receive FIFO Start Index Register (FR_RFSIR)*
- *Receive FIFO Start Data Offset Register (FR_RFSDOR)*
- *Receive FIFO Depth and Size Register (RFDSR)*
- *Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)*
- *Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)*
- *Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)*
- *Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)*
- *Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)*

29.6.3.7.2 Receive FIFO Control Data

The application can access the FIFOs at any time using the control bits in the following registers:

- *Global Interrupt Flag and Enable Register (FR_GIFER)*
- *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*

29.6.3.7.3 Receive FIFO Status Data

The current status of the receive fifo is provided in the following register:

- *Global Interrupt Flag and Enable Register (FR_GIFER)*
- *Receive FIFO A Read Index Register (FR_RFARIR)*
- *Receive FIFO B Read Index Register (FR_RFBRIR)*
- *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*

29.6.4 Flexray Memory Area Layout

The CC supports a wide range of possible layouts for the flexray memory area. Two basic layout modes can be selected by the FIFO address mode bit FR_MCR[FAM].

29.6.4.1 Flexray Memory Area Layout (FR_MCR[FAM] = 0)

Figure 529 shows an example layout for the FIFO address mode FR_MCR[FAM]=0. In this mode, the following set of rules applies to the layout of the flexray memory area:

- The flexray memory area is one contiguous region.
- The flexray memory area size is maximum 64 Kbytes.
- The flexray memory area starts at a 16 byte boundary

The flexray memory area contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*.

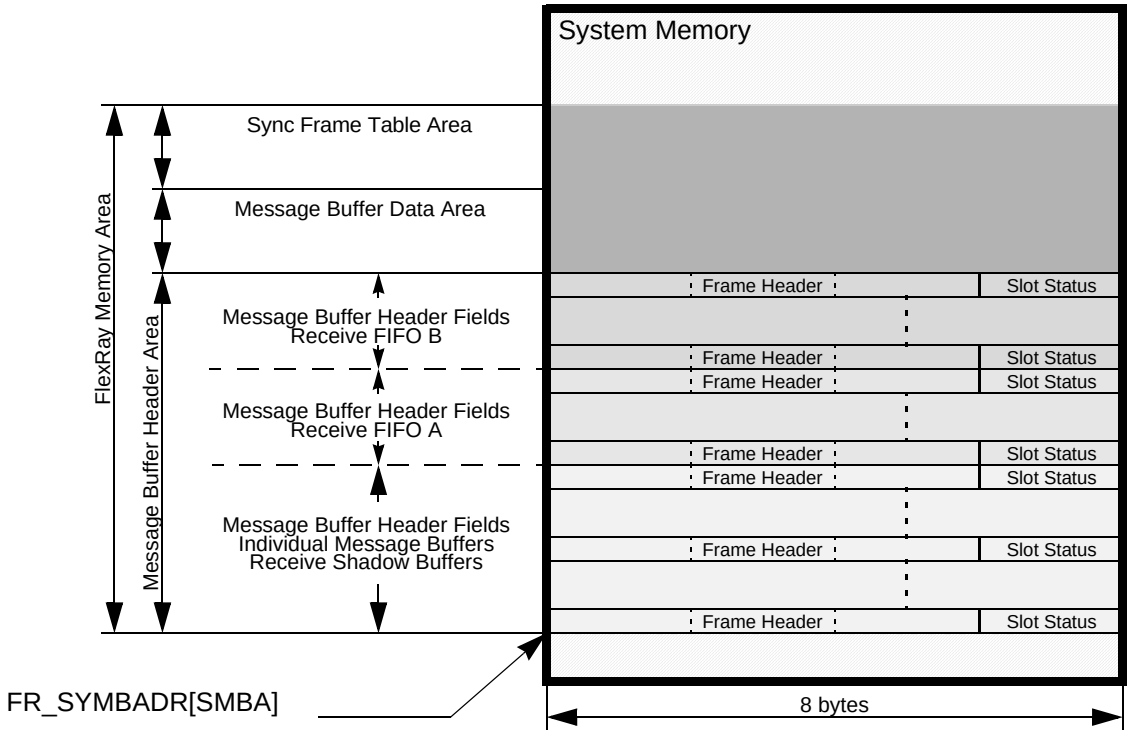


Figure 529. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 0)

29.6.4.2 FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

Figure 530 shows an example layout for the FIFO address mode FR_MCR[FAM]=1. The following set of rules applies to the layout of the flexray memory area:

- The flexray memory area consists of two contiguous regions.
- The size of each region is maximum 64 Kbytes.
- Each region start at a 16 byte boundary.

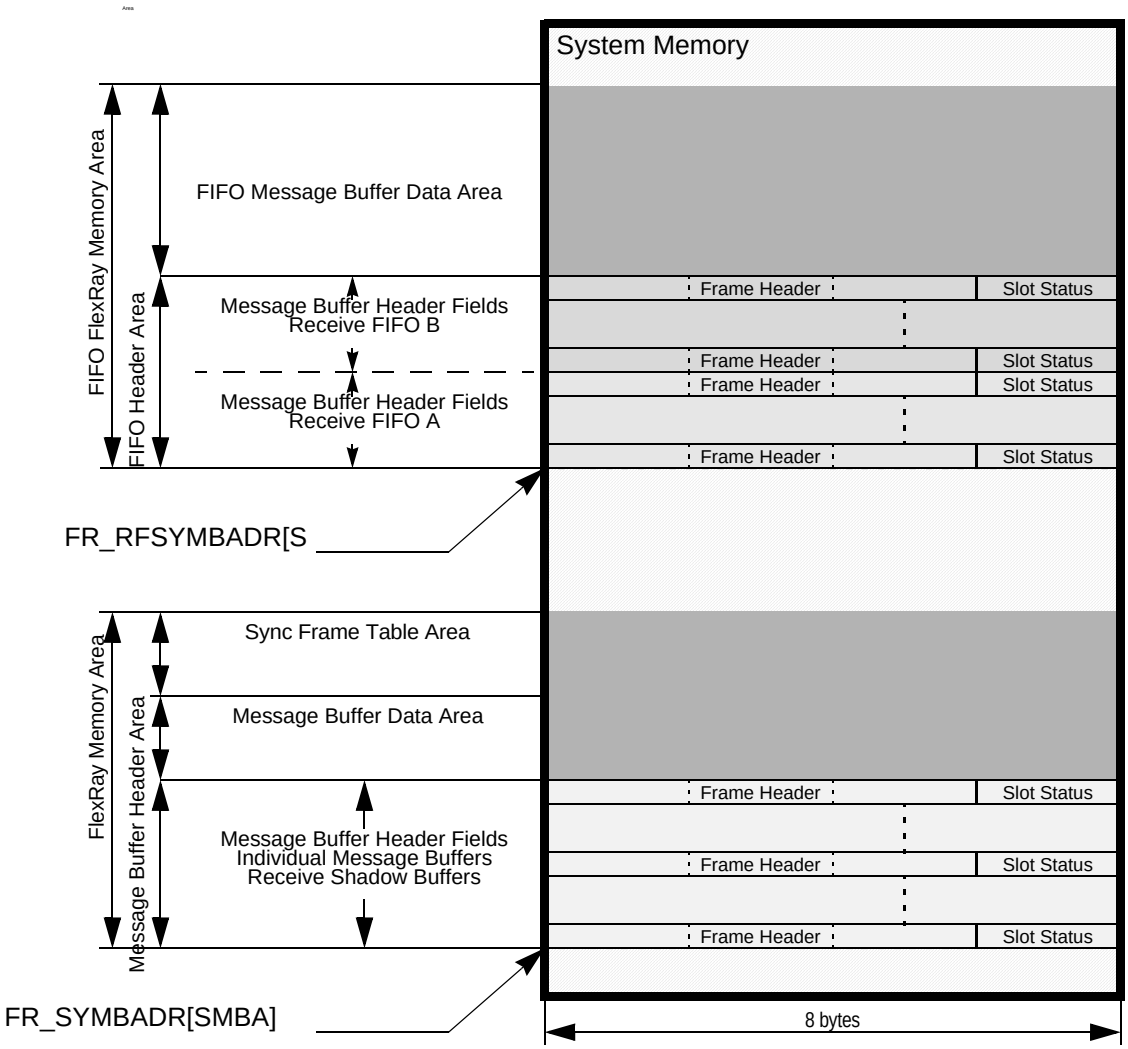


Figure 530. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

29.6.4.3 Message Buffer Header Area (FR_MCR[FAM] = 0)

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start byte address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill *Equation 27*.

Equation 27
SADR_MBHF = (i * 8) + FR_SYMBADR[SMBA]; (0 <= i <= 131)

2. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill *Equation 28*.

Equation 28
SADR_MBHF = (i * 8) + FR_SYMBADR[SMBA]; (0 <= i <= 1023)

Equation 29
SADR_MBHF = (i * 8) + FR_SYMBADR[SMBA]; (0 <= i <= 1023)

3. The message buffer header fields for each FIFO have to be a contiguous area.

29.6.4.4 Message Buffer Header Area (FR_MCR[FAM] = 1)

The message buffer header area contains all message buffer header fields of the physical message buffers for the individual message buffers and receiver shadow buffers. The following rules apply to the message buffer header fields for the two type of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill *Equation 30*.

Equation 30
SADR_MBHF = (i * 8) + FR_SYMBADR[SMBA]; (0 <= i <= 131)

29.6.4.5 FIFO Message Buffer Header Area (FR_MCR[FAM] = 1)

The FIFO message buffer header area contains all message buffer header fields of the physical message buffers for the FIFO. The following rules apply to the FIFO message buffer header fields.

1. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill *Equation 31*.

Equation 31
SADR_MBHF = (i * 8) + FR_RFSYMBADR[SMBA]; (0 <= i <= 1023)

2. The message buffer header fields for each FIFO have to be a contiguous area.

29.6.4.6 Message Buffer Data Area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

29.6.4.7 Sync Frame Table Area

The sync frame table area is used to provide a copy of the internal sync frame tables for application access. Refer to *Section 29.6.12: Sync Frame ID and Sync Frame Deviation Tables* for the description of the sync frame table area.

29.6.5 Physical Message Buffer Description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

29.6.5.1 Message Buffer Protection and Data Consistency

The physical message buffers are located in the flexray memory area. The CC provides no means to protect the flexray memory area from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

29.6.5.2 Message Buffer Header Field Description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in *Section 29.6.2.1: Message Buffer Header Field*. Each message buffer header field consists of two sections: the frame header section and the slot status section.

29.6.5.2.1 Frame Header Description

29.6.5.2.1.1 Frame Header Content

The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in *Figure 531*. A detailed description is given in *Table 457*.



Figure 531. Frame Header Structure (Receive Message Buffer and Receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in *Figure 532*. A detailed description is given in *Table 458*. The checks that will be performed are described in *Frame Header Checks*.

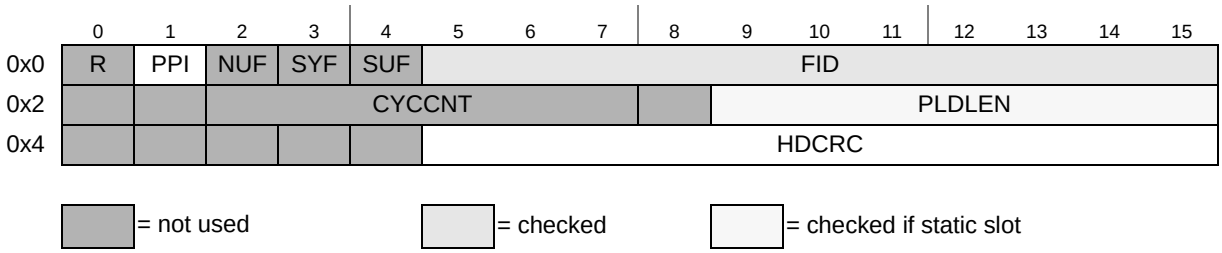


Figure 532. Frame Header Structure (Transmit Message Buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in *Figure 533*.

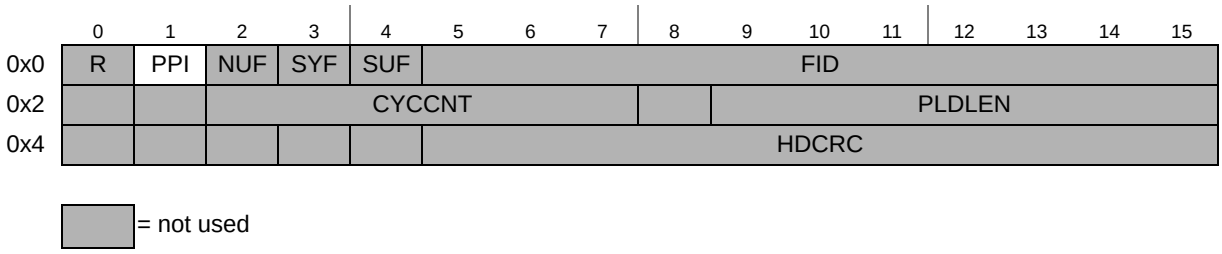


Figure 533. Frame Header Structure (Transmit Message Buffer for Key Slot)

29.6.5.2.1.2 Frame Header Access

The frame header is located in the flexray memory area. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 456*. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 456. Frame Header Write Access Constraints (Transmit Message Buffer)

Field	Static Segment	Dynamic Segment
FID	*POC:config* or MB_DIS	
PPI, PLDLEN, HDCRC	*POC:config* or MB_DIS or	MB_LCK

29.6.5.2.1.3 Frame Header Checks

As shown in *Figure 532* and *Figure 533* not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored, some are used for transmission, and some of them are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding *Message Buffer Frame ID Registers (FR_MBFIDRn)*. If the CC detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the *CHI Error Flag Register (FR_CHIERFR)*. The value of the FID field will be ignored and replaced by the value provided in the *Message Buffer Frame ID Registers (FR_MBFIDRn)*.

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the *Protocol Configuration Register 19 (FR_PCR19)*. If this is not fulfilled, the static payload length error flag SPL_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the *Protocol*

Configuration Register 24 (FR_PCR24). If this is not fulfilled, the dynamic payload length error flag DPL_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 457. Frame Header Field Descriptions (Receive Message Buffer and Receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 458. Frame Header Field Descriptions (Transmit Message Buffer)

Field	Description
R	**Reserved Bit** — This bit is not used, the value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used, the value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SYF	**Sync Frame Indicator** — This bit is not used, the value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SUF	**Startup Frame Indicator** — This bit is not used, the value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
FID	**Frame ID** — This field is checked as described in *Frame Header Checks*.
CYCCNT	**Cycle Count** — This field is not used, the value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.
PLDLEN	**Payload Length** — This field is checked and used as described in *Frame Header Checks*.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

29.6.5.2.2 Slot Status Description

The slot status is a read-only structure for the application and a write-only structure for the CC. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

29.6.5.2.2.1 Receive Message Buffer and Receive FIFO Slot Status Description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by *Table 459*.

Table 459. Receive Message Buffer Slot Status Content

Receive Message Buffer Type	Slot Status Content
Individual Receive Message Buffer assigned to both channels FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=1	see *Figure 534*
Individual Receive Message Buffer assigned to channel A FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB[=0	see *Figure 535*
Individual Receive Message Buffer assigned to channel B FR_MBCCFRn[CHA]=0 and FR_MBCCFRn[CHB]=1	see *Figure 536*
Receive FIFO Channel A Message Buffer	see *Figure 535*
Receive FIFO Channel B Message Buffer	see *Figure 536*

The meaning of the bits in the slot status structure is explained in *Table 460*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 534. Receive Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 535. Receive Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 536. Receive Message Buffer Slot Status Structure (ChB)

Table 460. Receive Message Buffer Slot Status Field Description

Field	Description
	Common Message Buffer Status Bits
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
CH	**Channel first valid received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all 1 first valid frame received on channel B
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1

Table 460. Receive Message Buffer Slot Status Field Description

Field	Description
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1

29.6.5.2.2.2 Transmit Message Buffer Slot Status Description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by *Table 461*.

Table 461. Transmit Message Buffer Slot Status Content

Transmit Message Buffer Type	Slot Status Content
Individual Transmit Message Buffer assigned to both channels FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=1	see *Figure 537*
Individual Transmit Message Buffer assigned to channel A FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=0	see *Figure 538*
Individual Transmit Message Buffer assigned to channel B FR_MBCCFRn[CHA]=0 and FR_MBCCFRn[CHB]=1	see *Figure 539*

The meaning of the bits in the slot status structure is described in *Table 460*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 537. Transmit Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 538. Transmit Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 539. Transmit Message Buffer Slot Status Structure (ChB)

Table 462. Transmit Message Buffer Slot Status Structure Field Descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1

Table 462. Transmit Message Buffer Slot Status Structure Field Descriptions (continued)

Field	Description
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

29.6.5.3 Message Buffer Data Field Description

The message buffer data field is used to store the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in *Table 463*. The structure of the message buffer data field is given in *Figure 540*.

Table 463. Message Buffer Data Field Minimum Length

physical message buffer assigned to	minimum length defined by
Individual Message Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive FIFO for channel A	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 0)
Receive FIFO for channel B	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 1)

Note: The CC will not access any locations outside the message buffer data field boundaries given by Table 463.

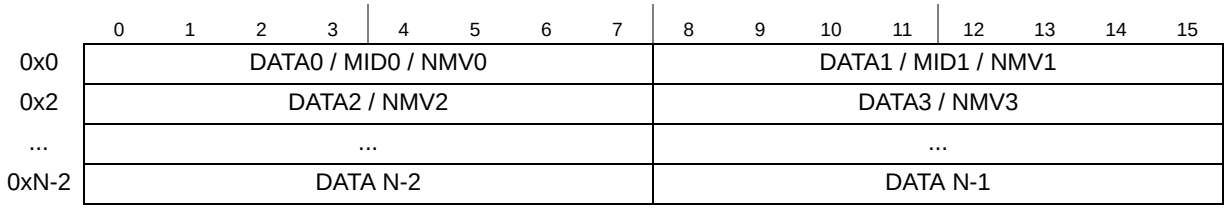


Figure 540. Message Buffer Data Field Structure

The message buffer data field is located in the flexray memory area; thus, the CC has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

29.6.5.3.1 Message Buffer Data Field Read Access

For transmit message buffers, the CC will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the *Message Buffer Index Registers (FR_MBIDXRn)*. While the message buffer is locked, the CC will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the *Receive FIFO A Read Index Register (FR_RFARIR)* or the *Receive FIFO B Read Index Register (FR_RFBRIR)* when the related fill levels in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)* indicate an non-empty FIFO.

29.6.5.3.2 Message Buffer Data Field Write Access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 464*.

Table 464. Frame Data Write Access Constraints

Field	CC/MB State
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK

Table 465. Frame Data Field Descriptions

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	Message Identifier — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	Network Management Vector — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. **Note:** The MID and NMV bytes replace the corresponding DATA bytes.

29.6.6 Individual Message Buffer Functional Description

The CC provides three basic types of individual message buffers:

1. Transmit Message Buffers
2. Receive Message Buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in *Section 29.6.3.4.1: Individual Message Buffer Configuration Data*.

29.6.6.1 Individual Message Buffer Configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the flexray memory area. The second step is the programming of the message buffer configuration registers, which is described in this section.

29.6.6.1.1 Common Configuration Data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the *Message Buffer Data Size Register (FR_MBDSR)*.

Depending on the current receive functionality of the CC, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the *Receive Shadow Buffer Index Register (FR_RSBIR)*.

29.6.6.1.2 Specific Configuration Data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled, i.e. FR_MBCCSRn[EDS] = 0

The individual message buffer type is defined by the MTD and MBT bits in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)* as given in *Table 466*.

Table 466. Individual Message Buffer Types

FR_MBCCSRn		Individual Message Buffer Description
MTD	MBT	
0	0	Receive Message Buffer
0	1	Reserved
1	0	Transmit Message Buffer
1	1	Reserved

The message buffer specific configuration data are

1. MTD bits in *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*
2. all fields and bits in *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*
3. all fields and bits in *Message Buffer Frame ID Registers (FR_MBFIDRn)*
4. all fields and bits in *Message Buffer Index Registers (FR_MBIDXRn)*

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions *Section 29.6.6.2: Transmit Message Buffers* and *Section 29.6.6.3: Receive Message Buffers*.

29.6.6.2 Transmit Message Buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A transmit message buffer is used by the application to provide message data to the CC that will be transmitted over the FlexRay Bus. The CC uses the transmit message buffers to provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a transmit message buffer by the following settings:

- FR_MBCCSRn[MBT] = 0 (single buffered message buffer)
- FR_MBCCSRn[MTD] = 1 (transmit message buffer)

29.6.6.2.1 Access Regions

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented, which is used to control the access to the data, control, and status bits of a message buffer. The access regions for transmit message buffers are depicted in *Figure 541*. A description of the regions is given in *Table 467*. If an region is active as indicated in *Table 468*, the access scheme given for that region applies to the message buffer.

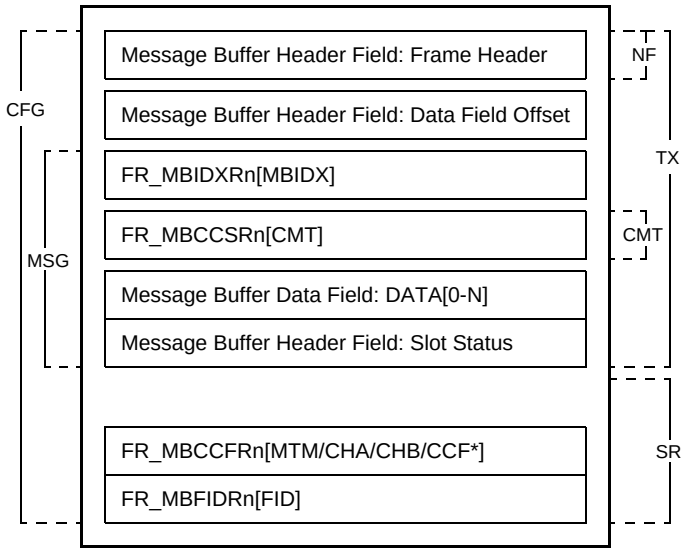


Figure 541. Transmit Message Buffer Access Regions

Table 467. Transmit Message Buffer Access Regions Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration
MSG	read/write	-	Message Data and Slot Status Access
NF	-	read-only	Message Header Access for Null Frame Transmission
TX	-	read/write	Message Transmission and Slot Status Update
CM	-	read-only	Message Buffer Validation
SR	-	read-only	Message Buffer Search

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT], and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC clear access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in *Figure 542*. A description of the states is given in *Table 468*, which also provides the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

29.6.6.2.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

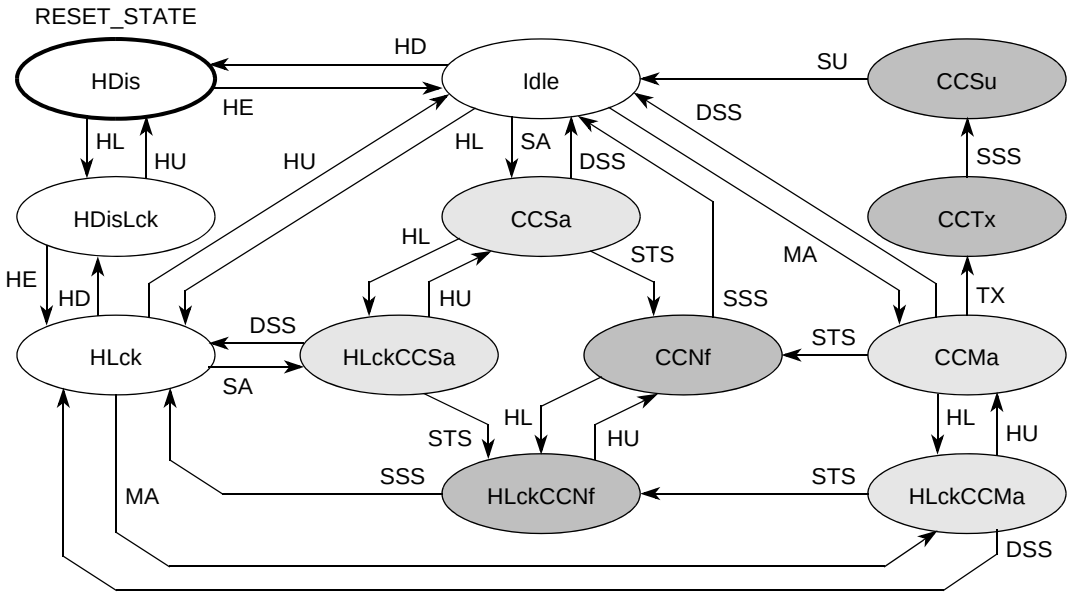


Figure 542. Transmit Message Buffer States

Table 468. Transmit Message Buffer State Description

State	FR_MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	CM, SR	**Idle** - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	–	**Disabled and Locked** - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	Locked - Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	–	Locked and Slot Assigned - Applications access to data, control, and status.Message buffer assigned to next static slot
CCNf	1	0	–	NF	Null Frame Transmission Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	Locked and Null Frame Transmission - Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	–	CM	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	–	Locked and Message Available - Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	–	TX	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index.

29.6.6.2.3 Message Buffer Transitions

29.6.6.2.3.1 Application Transitions

The application transitions can be triggered by the application using the commands described in *Table 469*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit FR_MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit FR_MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit FR_MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSRn[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set.

Table 469. Transmit Message Buffer Application Transitions

Transition	Command	Condition	Description
HE	FR_MBCCSRn[EDT]:= 1	FR_MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSRn[LCKT]:= 1	FR_MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

29.6.6.2.3.2 Module Transitions

The module transitions that can be triggered by the CC are described in *Table 470*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 470. Transmit Message Buffer Module Transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and FR_MBCCSRn[CMT] = 1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start. - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

29.6.6.2.3.3 Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 471*, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the

intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of *Table 471*.

Table 471. Transmit Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

29.6.6.2.4 Transmit Message Setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. The physical access to the message buffer data field is described in *Section 29.6.3.1: Individual Message Buffers*.

As indicated by *Table 468*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 469*. The state change is indicated through the FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the FR_MBCCSRn[DVAL] flag is negated.

29.6.6.2.5 Message Transmission

As a result of the message buffer search described in *Section 29.6.7: Individual Message Buffer Search*, the CC triggers the message available transition MA for up to two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The CC transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
2. the message data are still valid, i.e. FR_MBCCSRn[CMT] = 1

In this case, the CC triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in *Figure 543*. In this example, the message buffer with message buffer number n is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid, i.e. FR_MBCCSRn[CMT] = 1.

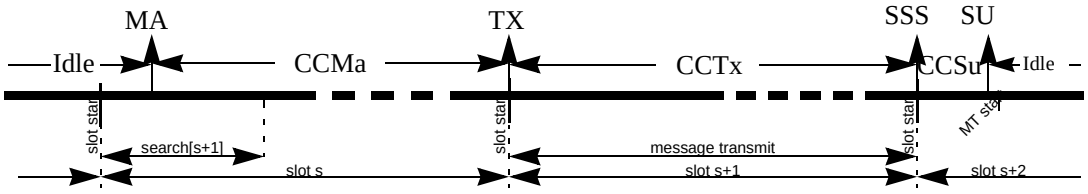


Figure 543. Message Transmission Timing

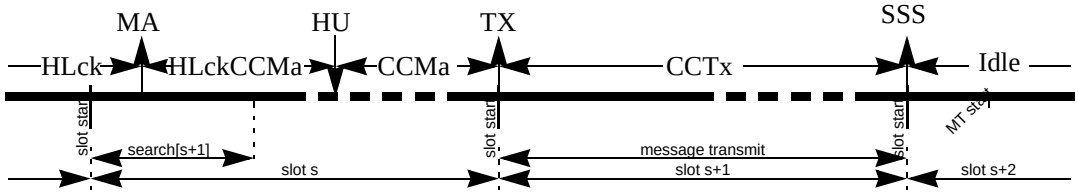


Figure 544. Message Transmission from HLck state with unlock

The amount of message data read from the flexray memory area and transferred to the FlexRay bus is determined by the following three items

1. the message buffer segment that the message buffer is assigned to, as defined by the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.
2. the message buffer data field size, as defined by the related field of the *Message Buffer Data Size Register (FR_MBDSR)*
3. the value of the PLDLEN field in the message buffer header field, as described in *Section 29.6.5.2.1: Frame Header Description*

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 * MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN <= MBSEG1DS, the CC reads and transfers 2*PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

29.6.6.2.6 Null Frame Transmission

A static slot with slot number S is assigned to the CC for channel A, if at least one transmit message buffer is configured with the FR_MBFIDRn[FID] set to S and FR_MBCCFRn[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the CC for channel A, and all transmit message buffers with FR_MBFIDRn[FID] = s and FR_MBCCFRn[CHA] = 1 are either not committed, i.e FR_MBCCSRn[CMT] = 0, or locked by the application, i.e. FR_MBCCSRn[LCKS] = 1, or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in *Section 29.6.7: Individual Message Buffer Search*, the CC triggers the slot assigned transition SA for up to two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission

in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in *Figure 545*.

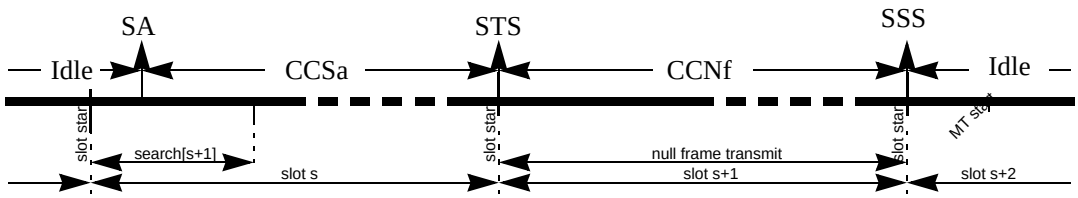


Figure 545. Null Frame Transmission from Idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in *Figure 546*.

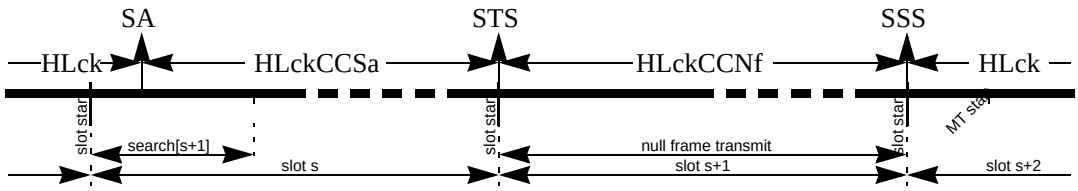


Figure 546. Null Frame Transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 547*.

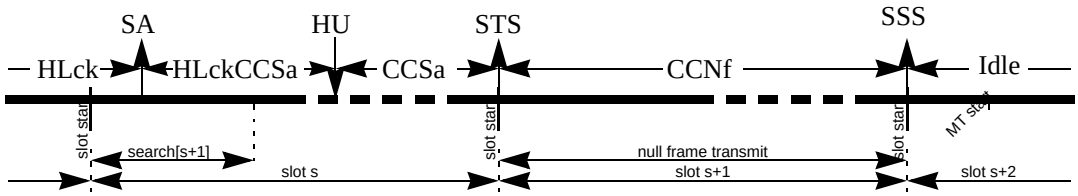


Figure 547. Null Frame Transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 548*.

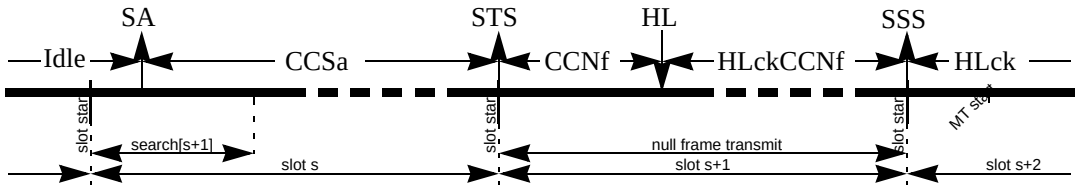


Figure 548. Null Frame Transmission from Idle state with locking

29.6.6.2.7 Message Buffer Status Update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

29.6.6.2.7.1 Message Buffer Status Update after Complete Message Transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the CC updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the CC sets the message buffer interrupt flag FR_MBCCSRn[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag FR_MBCCFRn[MTM], the CC changes the commit flag FR_MBCCSRn[CMT] and the valid flag FR_MBCCSRn[DVAL]. If the FR_MBCCFRn[MTM] flag is negated, the message buffer is in the *event transmission mode.* In this case, each committed message is transmitted only once. The commit flag FR_MBCCSRn[CMT] is cleared with the SU transition. If the FR_MBCCFRn[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The CC will not clear the FR_MBCCSRn[CMT] flag at the end of transmission and will set the valid flag FR_MBCCSRn[DVAL] to indicate that the message will be transmitted again.

29.6.6.2.7.2 Message Buffer Status Update after Incomplete Message Transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
2. The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the *Protocol Interrupt Flag Register 1 (FR_PIFR1)*.

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

29.6.6.2.7.3 Message Buffer Status Update after Null Frame Transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

29.6.6.3 Receive Message Buffers

The section provides a detailed description of the functionality of the receive message buffers. If receive message buffers are used it is required to configure the related receive shadow buffer as described in *Section 29.6.3.2: Receive Shadow Buffers*

A receive message buffer is used to receive a message from the FlexRay Bus based on individual filter criteria. The CC uses the receive message buffer to provide the following data to the application

1. message data received
2. information about the reception process
3. status information about the slot in which the message was received

A individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings

- FR_MBCCSRn[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented that is used to control the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are depicted in *Figure 549*. A description of the regions is given in *Table 472*. If an region is active as indicated in *Table 473*, the access scheme given for that region applies to the message buffer.

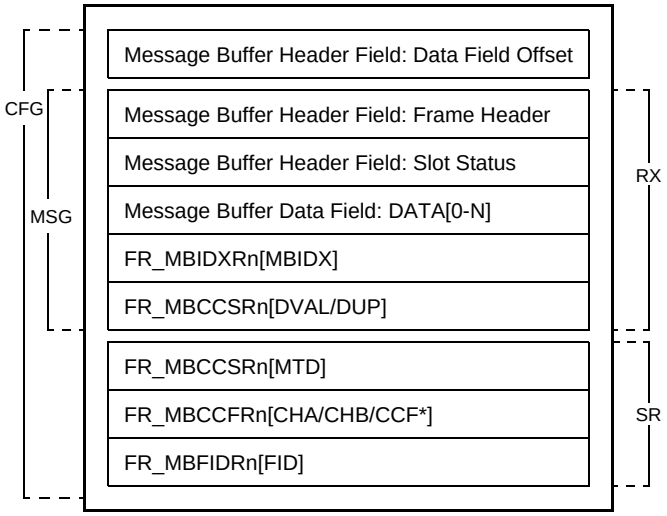


Figure 549. Receive Message Buffer Access Regions

Table 472. Receive Message Buffer Access Region Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	-	Message Data, Header, and Status Access
RX	-	write-only	Message Reception and Status Update
SR	-	read-only	Message Buffer Search Data

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT] and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC set access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in *Figure 550*. A description of the message buffer states is given in *Table 468*, which also provides the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

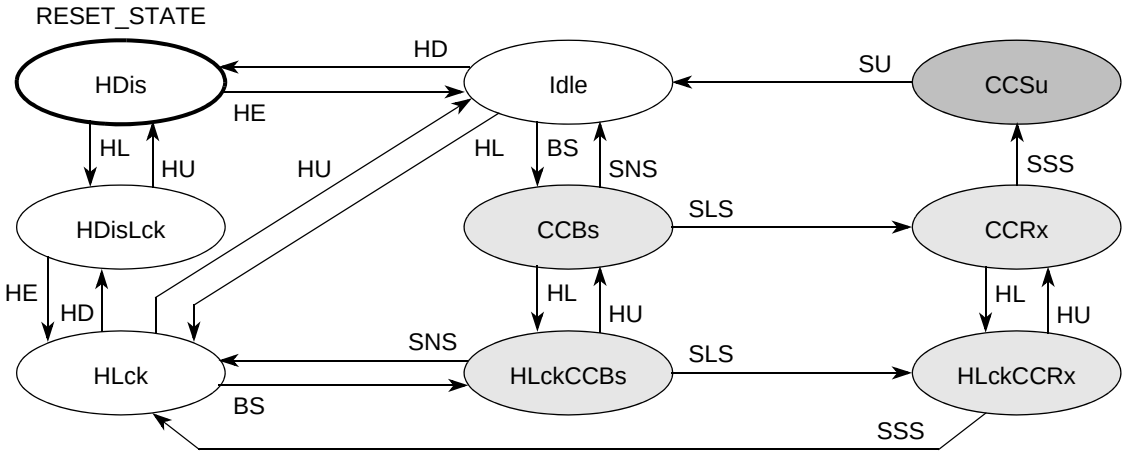


Figure 550. Receive Message Buffer States

Table 473. Receive Message Buffer States and Access

State	FR_MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	SR	**Idle** - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.

Table 473. Receive Message Buffer States and Access (continued)

State	FR_MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
HDisLck	0	1	CFG	–	**Disabled and Locked** - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	–	**Locked** - Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	–	–	**Buffer Subscribed** - Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	–	**Locked and Buffer Subscribed** - Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	–	–	**Message Receive** - Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	–	Locked and Message Receive - Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	–	RX	**Status Update** - Message buffer status update. Update of status flags, the slot status field, and the header index.

29.6.6.3.1 Message Buffer Transitions

29.6.6.3.1.1 Application Transitions

The application transitions that can be triggered by the application using the commands described in *Table 474*. The application issues the commands by writing to the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit FR_MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit FR_MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit FR_MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSRn[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *CHI Error Flag Register (FR_CHIERFR)* is set.

Table 474. Receive Message Buffer Application Transitions

Transition	Host Command	Condition	Description
HE	FR_MBCCSRn[EDT]:= 1	FR_MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSRn[LCKT]:= 1	FR_MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

29.6.6.3.1.2 Module Transitions

The module transitions that can be triggered by the CC are described in *Table 475*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 475. Receive Message Buffer Module Transitions

Transition	Condition	Description
BS	slot match and CycleCounter match	Buffer Subscribed - The message buffer filter matches next slot and cycle.
SLS	slot start	Slot Start - Start of either Static Slot or Dynamic Slot.
SNS	symbol window start or NIT start	Symbol Window or NIT Start - Start of either Symbol Window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	status updated	Status Updated - Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

29.6.6.3.1.3 Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in *Table 476*, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 476. Receive Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

29.6.6.3.2 Message Reception

As a result of the message buffer search, the CC changes the state of up to two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see *Section 29.6.6.3.5: Receive Shadow Buffers Concept*. The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

29.6.6.3.3 Message Buffer Update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the *CHI Error Flag Register (FR_CHIERFR)*.

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in *Table 477*.

Table 477. Receive Message Buffer Update

vSS!ValidFrame	vRF!Header!NFIndicator	Update description
1	1	Valid non-null frame received. - Message Buffer Data Field updated. - Frame Header Field updated. - Slot Status Field updated. - DUP:= 1 - DVAL:= 1 - MBIF:= 1
1	0	Valid null frame received. - Message Buffer Data Field *not* updated. - Frame Header Field *not* updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed - MBIF:= 1

Table 477. Receive Message Buffer Update (continued)

vSS!ValidFrame	vRF!Header!NFIndicator	Update description
0	x	No valid frame received. - Message Buffer Data Field not updated. - Frame Header Field not updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed. - MBIF:= 1, if the slot was not an empty dynamic slot. **Note:** An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

Note: If the number of the last slot in the current communication cycle on a given channel is n, then all receive message buffers assigned to this channel with FR_MBFIDRn[FID] > n will not be updated at all.

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in *Figure 551*.

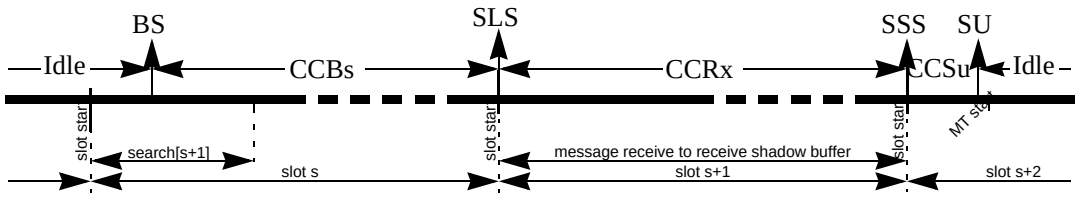


Figure 551. Message Reception Timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following two items:

1. the message buffer segment that the message buffer is assigned to, as defined by the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*.
2. the message buffer data field size, as defined by the related field of the *Message Buffer Data Size Register (FR_MBDSR)*
3. the number of bytes received over the FlexRay bus

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2*FR_MBDSR.MBSEG1DS, the CC writes only 2*FR_MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2*FR_MBDSR.MBSEG1DS, the CC writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with FR_MBDSR.MBSEG2DS.

29.6.6.3.4 Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message

buffer. The access to the message buffer data field is described in *Section 29.6.3.1: Individual Message Buffers*.

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the CC will not change the content of the message buffer.

29.6.6.3.5 Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in *Section 29.6.3.2: Receive Shadow Buffers*.

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see *Table 477*), the CC writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the *Message Buffer Index Registers (FR_MBIDXRn)* with the content of the corresponding internal shadow buffer index register. In all other cases, the CC writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the flexray memory area located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the flexray memory area accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the *Message Buffer Index Registers (FR_MBIDXRn)*. Instead, the index of the message buffer header field must be fetched from the *Message Buffer Index Registers (FR_MBIDXRn)*.

29.6.7 Individual Message Buffer Search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
2. slot start in the static segment
3. minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n+1*.

In general, the message buffer search for the next slot *n* considers only message buffers which are

1. enabled, i.e. FR_MBCCSRn[EDS] = 1, and
2. matches the next slot *n*, i.e. FR_MBFIDRn[FID] = *n*, and

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of *Table 478*. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of *Table 479*. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to *Table 478* for the static segment and according to *Table 479* for the dynamic segment are selected.

Table 478. Message Buffer Search Priority (static segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	1	-	0	1	transmit buffer, matches cycle count, not committed	SA
	1	1	-	1	transmit buffer, matches cycle count, locked	SA
2	1	-	-	-	transmit buffer	SA
3	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section 29.6.7.1: Message Buffer Cycle Counter Filtering*

Table 479. Message Buffer Search Priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section 29.6.7.1: Message Buffer Cycle Counter Filtering*

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in *Section 29.6.7.2: Message Buffer Channel Assignment Consistency*.

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in *Section 29.6.3.1: Individual Message Buffers*.

29.6.7.1 Message Buffer Cycle Counter Filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*. This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled, i.e. CCFE = 0, this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

Equation 32 MBCCFRn[CCFE] = 0

Equation 33

CYCCNT & MBCCFRn[CCFMSK] = MBCCFRn[CCFVAL] & MBCCFRn[CCFMSK]

29.6.7.2 Message Buffer Channel Assignment Consistency

The message buffer channel assignment given by the CHA and CHB bits in the *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)* defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if FR_MBCCFRn[CHA] = 1 and transmits or receives on channel B if FR_MBCCFRn[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is depicted in *Figure 552*.

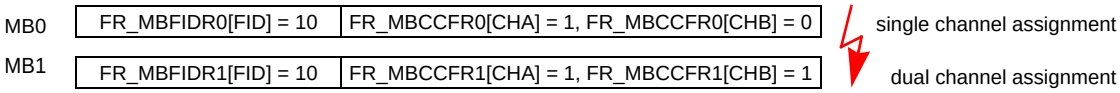


Figure 552. Inconsistent Channel Assignment

29.6.7.3 Node Related Slot Multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to *Table 479* the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

29.6.7.4 Message Buffer Search Error

There are two kinds of errors which may occur during message buffer search[(n)].

n. The FIFO reception is not affected by the search errors. Additionally, if no rx buffer has been found due to an search error, the received frame is considered for FIFO reception.

29.6.7.4.1 Message Buffer Search Start while Running

If the message buffer search is running in slot *n-1* and the next message buffer search start event appears due to the start of slot *n*, the message buffer search engine is stopped and the Message Buffer Search Error Flag MBS_EF is set in the *CHI Error Flag Register (FR_CHIERFR)*. As a result of this stop, no individual message buffer is identified for transmission or reception in slot *n*. Additionally, the search engine will not be started in slot *n*, and consequently no individual message buffer is identified for transmission or reception in slot *n+1*.

A message buffer search error appears only if the CHI frequency is too slow to allow the search through all message buffers to be completed within the NIT or a minislot.

For more details of minimum required CHI frequency see *Section 29.7.5: Number of Usable Message Buffers*.

29.6.7.4.2 Illegal Message Buffer Index Found

If the message buffer search has finished the message buffer search in slot *n-1*, it retrieves the data offset values for the found message buffers and the receive shadow buffers. If one of these message buffers contains an illegal message buffer index, the Message Buffer Search Error Flag MBS_EF is set in the *CHI Error Flag Register (FR_CHIERFR)* is set and no individual message buffer is identified for transmission or reception in slot *n*. The legal message buffer index values for the individual and receive shadow buffers are specified in *Section 29.5.2.52: Receive Shadow Buffer Index Register (FR_RSBIR)* and *Section 29.5.2.82: Message Buffer Index Registers (FR_MBIDXRn)*.

29.6.8 Individual Message Buffer Reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on *Specific Configuration Data*. The common configuration data given in the section on *Specific Configuration Data* can not be reconfigured when the protocol is out of the *POC:config* state.

29.6.8.1 Reconfiguration Schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

29.6.8.1.1 Basic Type Not Changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if the message buffer transfer direction bit FR_MBCCSRn[MTD] are not changed. This type of reconfiguration is denoted by RC1 in *Figure 553*. Transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state.

29.6.8.1.2 Buffer Type Not Changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer. This type of reconfiguration is denoted by RC2 in *Figure 553*. It applies to transmit and receive message buffers. Transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

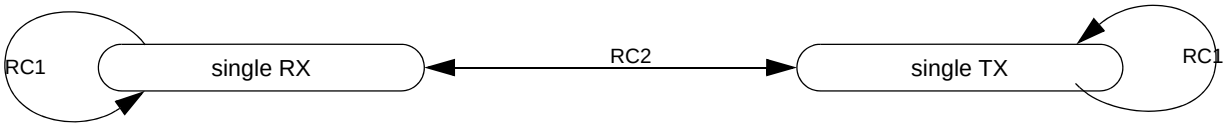


Figure 553. Message Buffer Reconfiguration Scheme

29.6.9 Receive FIFOs

This section provides the functional description of the two receive FIFOs.

29.6.9.1 Overview

The two receive FIFOs implement the queued message buffer concept defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.* One FIFO is assigned to channel A, the other FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in *Section 29.6.3.3: Receive FIFO*. The area in the flexray memory area for each of the two FIFOs is characterized by:

- The FIFO system memory base address
- The index of the first FIFO entry given by *Receive FIFO Start Index Register (FR_RFSIR)*
- The data field offset of the data field belonging to the first FIFO entry given by *Receive FIFO Start Data Offset Register (FR_RFSDOR)*
- The number of FIFO entries and the length of each FIFO entry as given by *Receive FIFO Depth and Size Register (RFDSR)*

29.6.9.2 FIFO Configuration

The FIFOs can be configured for two different locations of the system memory base address via the FIFO address mode bit FAM in the *Module Configuration Register (FR_MCR)*.

29.6.9.2.1 Single System Memory Base Address Mode

This mode is configured, when the FIFO address mode flag FR_MCR[FAM] is set to 0. In this mode, the location of the system memory base address for the FIFO buffers is *System Memory Base Address Register (FR_SYMBADR)*.

29.6.9.2.2 Dual System Memory Base Address Mode

This mode is configured, when the FIFO address mode flag FR_MCR[FAM] is set to 1. In this mode, the location of the system memory base address for the FIFO buffers is *Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)*.

The FIFO control and configuration data are given in *Section 29.6.3.7: Receive FIFO Control and Configuration Data*. The configuration of the FIFOs consists of two steps.

The first step is the allocation of the required amount of flexray memory area for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see *Section 29.6.4: Flexray Memory Area Layout*.

The second step is the programming of the configuration data register while the PE is in *POC:config.*

The following steps configure the layout of the FIFO.

- Configure the FIFO update and address modes in *Module Configuration Register (FR_MCR)*
- Configure the FIFO system memory base address
- Configure the *Receive FIFO Start Index Register (FR_RFSIR)* with the first message buffer header index that belongs to the FIFO
- Configure the *Receive FIFO Start Data Offset Register (FR_RFSDOR)* with the data field offset of the data field belonging to the first message buffer that belongs to the FIFO
- Configure the *Receive FIFO Depth and Size Register (RFDSR)* with FIFO entry size
- Configure the *Receive FIFO Depth and Size Register (RFDSR)* with FIFO depth
- Configure the FIFO Filters

29.6.9.3 FIFO Periodic Timer

The FIFO periodic timer is used to generate an FIFO almost-full interrupt at certain point in time, if the almost-full watermark is not reached, but the FIFO is not empty. This can be used to prevent frames from get stuck in the FIFO for a long time.

The FIFO periodic timer is configured via the *Receive FIFO Periodic Timer Register (FR_RFPTR)*. If the periodic timer duration FR_RFPTR[PTD] is configured to 0x0000, the periodic timer is continuously expired. If the periodic timer duration FR_RFPTR[PTD] is configured to 0x3FFF, the periodic timer never expires. If the periodic timer is configured to a value *ptd,* greater than 0x0000 and smaller 0x3FFF, the periodic timer expires and is restarted at the start of every communication cycle, and expires and is restarted after *ptd* macroticks have been elapsed.

29.6.9.4 FIFO Reception

The FIFO reception is a CC internal operation.

A message frame reception is directed into the FIFO, if no individual message buffer is assigned for transmission or subscribed for reception for the current slot. In this case the FIFO filter path shown in *Figure 554* is activated.

If the FIFO filter path indicates that the received frame has to be appended to the FIFO and the FIFO is not full, the CC writes the received frame header into the message buffer header field indicated by the CC internal FIFO write index. The frame payload data are written into the corresponding message buffer data field. If the status of the received frame indicates a valid non-null frame, the slot status information is written into the message buffer header field and the CC internal FIFO write index is updated by 1 and the fifo fill level FLA (FLB) in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)* is incremented.If the status of the received frame indicates an invalid or null frame, the frame is not appended to the FIFO.

29.6.9.5 FIFO Almost-Full Interrupt Generation

If the fifo fill level FLA (FLB) is updated after a frame reception and exceeds the FIFO watermark level WM, i.e. FLA>WM_A (FLB>WM_B), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.If the periodic timer expires, and

FIFOA (FIFOB) is not empty, i.e. FLA>0 (FLB>0), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.

29.6.9.6 FIFO Overflow Error Generation

If the FIFOA (FIFOB) is full, i.e. FLA=FIFO_DEPTH$_A$ (FLB=FIFO_DEPTH$_B$) and the conditions for a FIFO reception as described in *Section 29.6.9.4: FIFO Reception* are fulfilled, then the fifo overflow error flag FR_CHIERFR[FOVA_EF] (FR_CHIERFR[FOVB_EF]) is asserted.

29.6.9.7 FIFO Message Access

The FIFOA (FIFOB) contains valid messages if the FIFO fill level given in the fields FLA (FLB) in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)* is greater than 0. The *Receive FIFO A Read Index Register (FR_RFARIR)* and the (*Receive FIFO B Read Index Register (FR_RFBRIR)*) point to a message buffer with valid content and the oldest frames stored in the FIFO.The respective read data field offsets can be calculated according to *Equation 26*.

If the FIFO fill level FLA (FLB) in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)* is 0, than the FIFOA (FIFOB) contains no valid messages and the corresponding read index register *Receive FIFO A Read Index Register (FR_RFARIR)* or (*Receive FIFO B Read Index Register (FR_RFBRIR)* point to a message buffer with invalid content. In this case the application must not read data from this FIFO.

To access the oldest message in the FIFOA (FIFOB), the application first reads the read index RDIDX out of the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*). This read index points to the message buffer header field of the oldest message buffer that contains valid received message data. The data field offset belonging to this message buffer must be calculated by the application according to *Equation 26*. The application can access the message data as described in *Section 29.6.3.3: Receive FIFO*. When the application has read the message buffer data and status information, it can update the FIFO as described in *Section 29.6.9.8: FIFO Update*.

29.6.9.8 FIFO Update

The application updates the FIFOA (FIFOB) by writing a pop count value *pc* different from 0 to the PCA (PCB) field in the *Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)*.

As a result of the this operation, the CC removes the oldest *pc* entries from FIFOA (FIFOB).

If the specified pop count value *pc* is greater than the current fill level *fl* provided in FLA (FAB) field, then only *fl* entries are removed from the FIFOA (FIFOB), the remaining *fl*-*pc* requested pop operations are discarded without any notification. In this case FIFOA (FIFOB) is empty after the update operation.

The read index in the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) is incremented by the number of removed items. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index defined in *Receive FIFO Start Index Register (FR_RFSIR)* automatically.

29.6.9.8.1 FIFO Interrupt Flag Update

The FIFO Interrupt Flag Update mode is configured, when the FIFO update mode flag FR_MCR[FUM] is set to 0. In this mode FIFOA (FIFOB) will be updated by 1 entry, when the interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is written with 1 by the application.

If the FIFO is empty, the update request is ignored without any notification.

The read index in the *Receive FIFO A Read Index Register (FR_RFARIR)* (*Receive FIFO B Read Index Register (FR_RFBRIR)*) is incremented by 1, if the FIFO was not empty. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index automatically.

29.6.9.9 FIFO Filtering

The FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The CC provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is depicted in *Figure 554*.

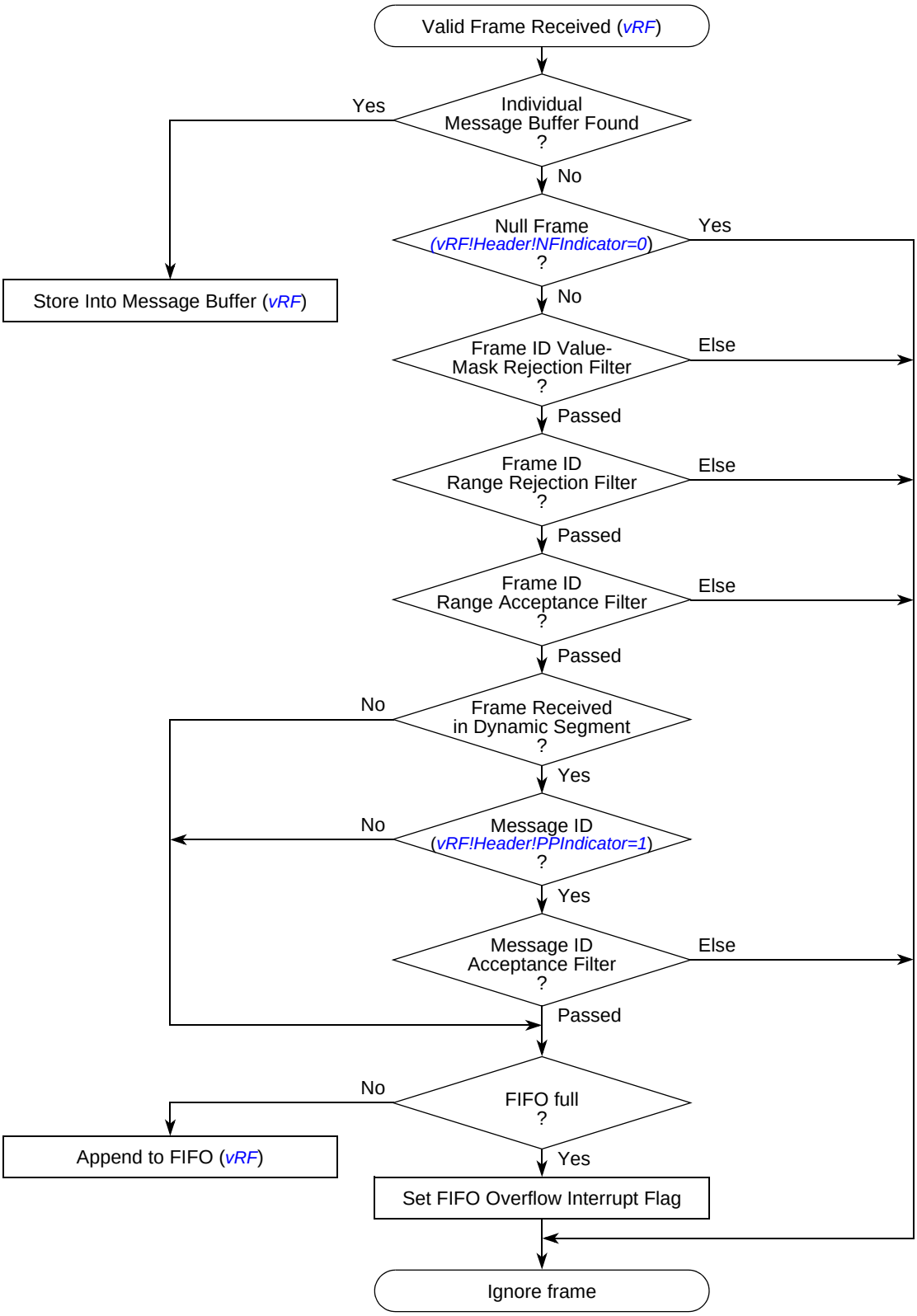


Figure 554. Received Frame FIFO Filter Path

A received frame passes the FIFO filtering if it has passed all three type of filter.

29.6.9.9.1 RX FIFO Frame ID Value-Mask Rejection Filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the *Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)* and the *Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)*. Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e. is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if *Equation 34* is fulfilled.

Equation 34

$$FID \,\&\, FR_RFFIDRFMR[FIDRFMSK] \neq$$
$$\neq FR_RFFIDRFVR[FIDRFVAL] \,\&\, FR_RFFIDRFMR[FIDRFMSK]$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- FR_RFFIDRFVR[FIDRFVAL]:= 0x000 and FR_RFFIDRFMR[FIDRFMSK]:= 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- FR_RFFIDRFMR[FIDRFMSK]:= 0x000

Using the settings above, *Equation 34* can never be fulfilled (0!= 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

29.6.9.9.2 RX FIFO Frame ID Range Rejection Filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the *Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)* and controlled by the *Receive FIFO Range Filter Control Register (FR_RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e. FR_RFRFCTR[FiMD] = 1 and FR_RFRFCTR[FiEN] = 1, *Equation 35* is fulfilled.

Equation 35

$$(FID < FR_RFRFCFR_{SEL}[SID_{IBD=0}]) \text{ or } (FR_RFRFCFR_{SEL}[SID_{IBD=1}] < FID)$$

Consequently, all frames with a frame ID that fulfills *Equation 36* for at least one of the enabled rejection filters will be rejected and thus not pass.

Equation 36

$$FR_RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq FR_RFRFCFR_{SEL}[SID_{IBD=1}]$$

29.6.9.9.3 RX FIFO Frame ID Range Acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the *Receive FIFO Range Filter Configuration Register*

(FR_RFRFCFR) and controlled by the *Receive FIFO Range Filter Control Register (FR_RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, i.e. FR_RFRFCTR[FiMD] = 0 and FR_RFRFCTR[FiEN] = 1, *Equation 37* is fulfilled.

Equation 37

$$FR_RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq FR_RFRFCFR_{SEL}[SID_{IBD=1}]$$

29.6.9.9.4 RX FIFO Message ID Acceptance Filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the *Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)* and the *Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)*. This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if *Equation 38* is fulfilled.

Equation 38

$$MID \,\&\, FR_RFMIDAFMR[MIDAFMSK] =$$

$$= FR_RFMIDAFMR[MIDAFVAL] \,\&\, FR_RFMIDAFMR[MIDAFMSK]$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- FR_RFMIDAFMR[MIDAFMSK]:= 0x000

Using the settings above, *Equation 38* is always fulfilled and all frames will pass.

29.6.10 Channel Device Modes

This section describes the two FlexRay channel device modes that are supported by the CC.

29.6.10.1 Dual Channel Device Mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX, and FR_A_TX_EN is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and FR_B_TX_EN is connected to the physical bus channel B. The dual channel system is shown in *Figure 555*.

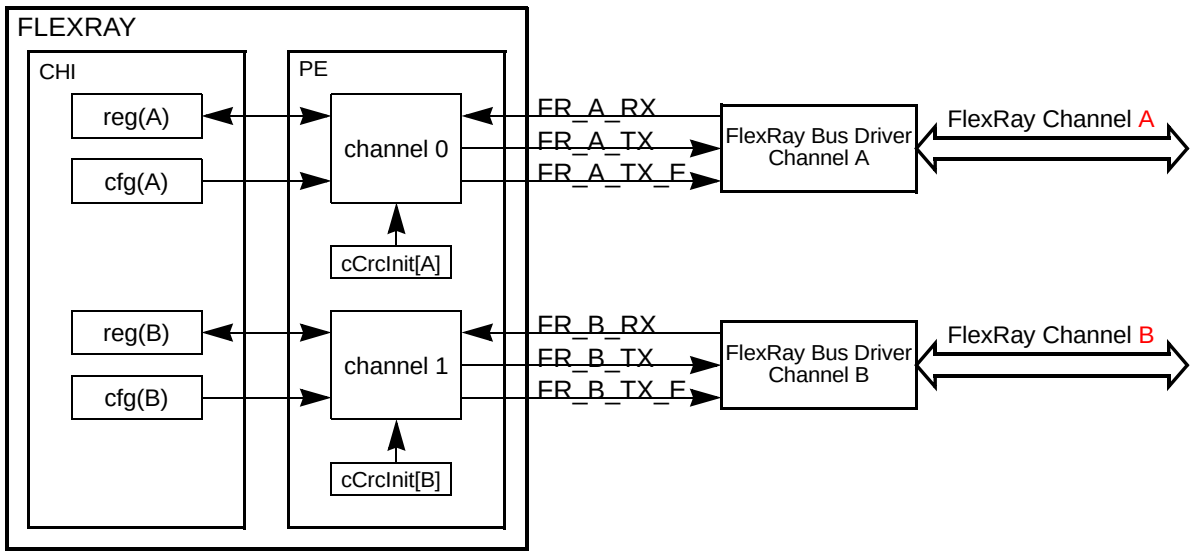


Figure 555. Dual Channel Device Mode

29.6.10.2 Single Channel Device Mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and FR_A_TX_EN and can be connected to either the physical bus channel A (shown in *Figure 556*) or the physical bus channel B (shown in *Figure 557*).

If the device is configured as a single channel device by setting FR_MCR[SCM] to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of FR_MCR[CHA] and FR_MCR[CHB], the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit FR_MCR[CHA] must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit FR_MCR[CHB] must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

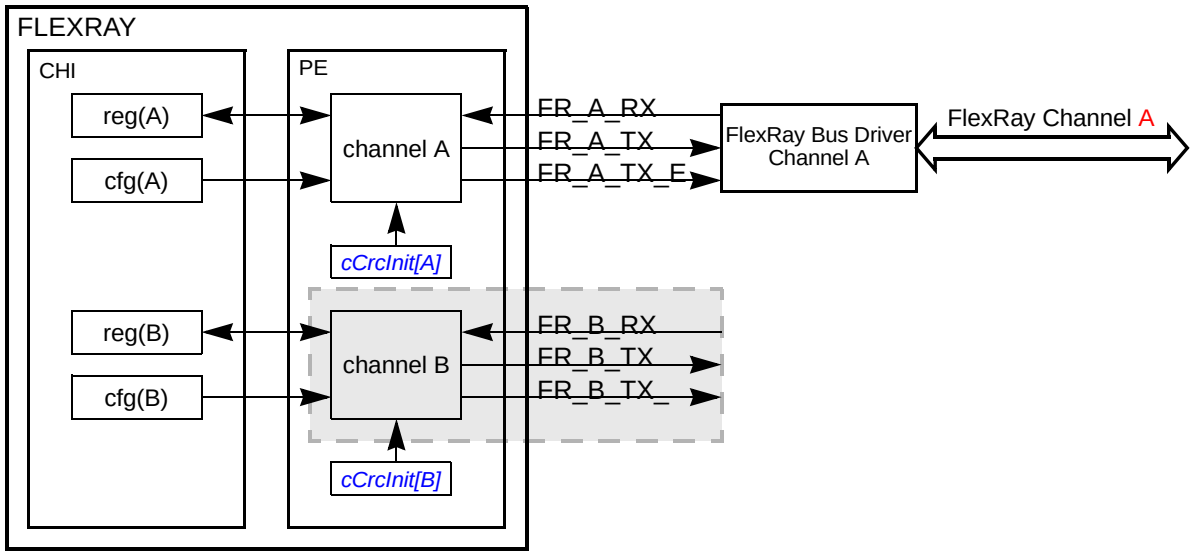


Figure 556. Single Channel Device Mode (Channel A)

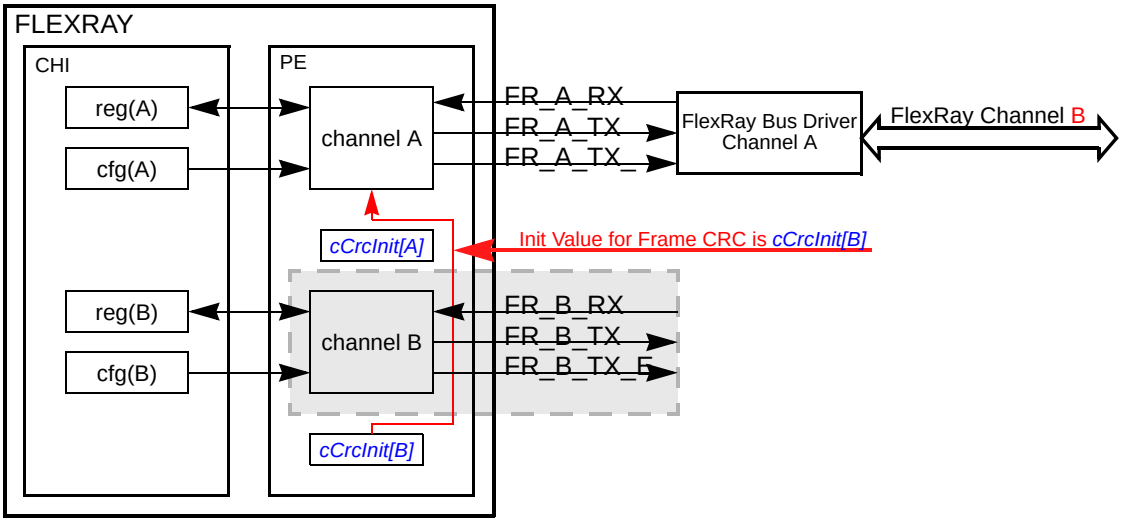


Figure 557. Single Channel Device Mode (Channel B)

29.6.11 External Clock Synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the *Protocol Operation Control Register (FR_POCR)*. The PE applies the external correction values in the next even-odd cycle pair as shown in *Figure 558* and *Figure 559*.

Note: The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2n+1 shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle 2n and before the end of the static segment of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle 2n+1. If the value is not applied in cycle 2n+1, then the value will be applied in the cycle 2n+3. Refer to *Figure 558* for timing details.

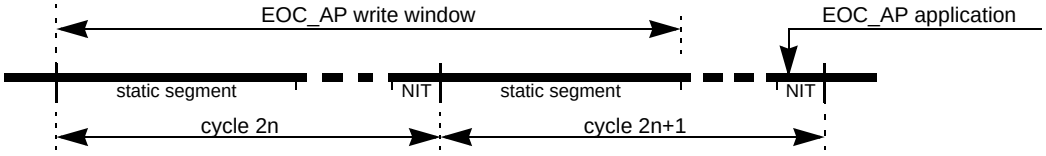


Figure 558. External Offset Correction Write and Application Timing

If the rate correction for the cycle pair [2n+2, 2n+3] shall be affect by the external offset correction, the ERC_AP field must be written to after the start of cycle 2n and before the end of the static segment start of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle pair [2n+2, 2n+3]. If the value is not applied for cycle pair [2n+2, 2n+3], then the value will be applied for cycle pair [2n+4, 2n+5]. Refer to *Figure 559* for details.

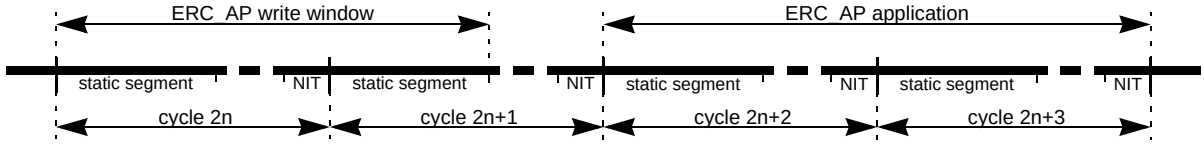


Figure 559. External Rate Correction Write and Application Timing

29.6.12 Sync Frame ID and Sync Frame Deviation Tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The CC provides the means to write a copy of these internal tables into the flexray memory area and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the CC will not overwrite these tables and the application can read a consistent snapshot.

Note: Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

29.6.12.1 Sync Frame ID Table Content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

29.6.12.2 Sync Frame Deviation Table Content

The Sync Frame Deviation Table is a snapshot of the protocol related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

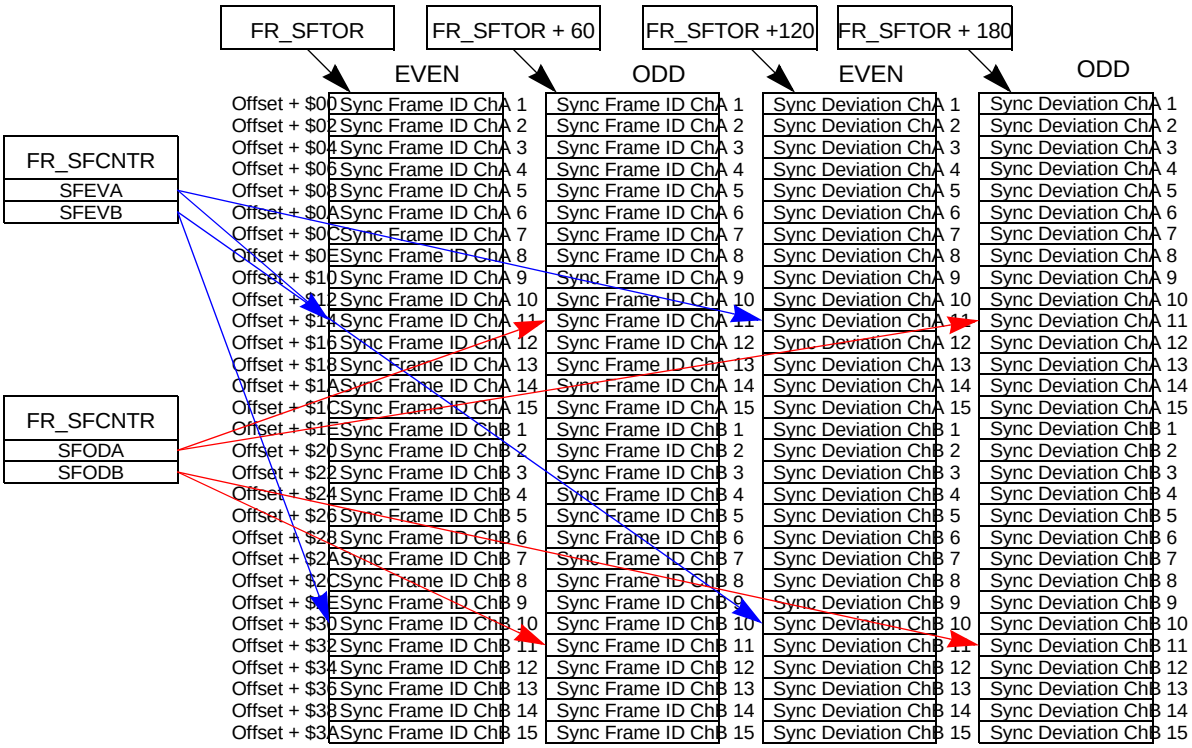


Figure 560. Sync Table Memory Layout

29.6.12.3 Sync Frame ID and Sync Frame Deviation Table Setup

The CC writes a copy of the internal synchronization frame ID and deviation tables into the flexray memory area if requested by the application. The application must provide the appropriate amount of flexray memory area for the tables. The memory layout of the tables is given in *Figure 560*. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the *Sync Frame Table Offset Register (FR_SFTOR)*.

29.6.12.4 Sync Frame ID and Sync Frame Deviation Table Generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the flexray memory area using the *Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)*. A summary of the copy modes is given in *Table 480*.

Table 480. Sync Frame Table Generation Modes

FR_SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
1	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting FR_SFTCCSR[SIDEN] to 1, the CC starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The CC checks if the application has locked the tables by reading the FR_SFTCCSR[ELKS] lock status bit. If this bit is set, the CC will not update the table in this cycle. If this bit is cleared, the CC locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the CC clears the even table valid status bit FR_SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the flexray memory area, the CC sets the even table valid bit FR_SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the *Protocol Interrupt Flag Register 1 (FR_PIFR1)*. If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the CC from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger FR_SFTCCSR[ELKT]. When the even table is not currently updated by the CC, the lock is granted and the even table lock status bit FR_SFTCCSR[ELKS] is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the Sync Frame Counter Register (FR_SFCNTR). The value in the FR_SFTCCSR[CYCNUM] field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the FR_SFCNTR[SFEVA] and FR_SFCNTR[SFEVB] fields. The application can now start to read the sync table data from the locations given in *Figure 560*.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger FR_SFTCCSR[ELKT] again. The even table lock status bit FR_SFTCCSR[ELKS] is reset immediately.

If the sync frame table generation is disabled, the table valid bits FR_SFTCCSR[EVAL] and FR_SFTCCSR[EVAL] are reset when the counter values in the Sync Frame Counter Register (FR_SFCNTR) are updated. This is done because the tables stored in the flexray memory area are no longer related to the values in the Sync Frame Counter Register (FR_SFCNTR).

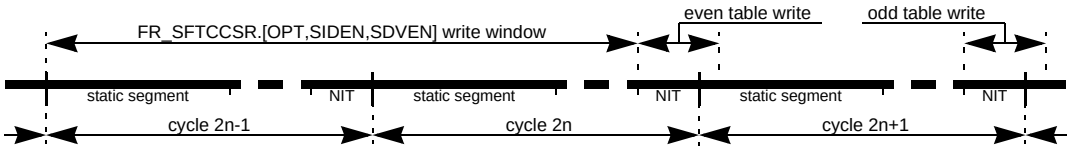


Figure 561. Sync Frame Table Trigger and Generation Timing

29.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the flexray memory area during the table write windows shown in *Figure 561*. During the table write, the application can not lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

29.6.12.5.1 Sync Frame Table Locking and Unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the *Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)*. If the affected table is not currently written to the flexray memory area, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the flexray memory area, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

29.6.13 MTS Generation

The CC provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the *MTS A Configuration Register (FR_MTSACFR)* and *MTS B Configuration Register (MTSBCFR)*.

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the *MTS A Configuration Register (FR_MTSACFR)* or *MTS B Configuration Register (MTSBCFR)*. If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if *Equation 40*, *Equation 41*, and *Equation 42* are fulfilled.

Equation 39 FR_PSR0[PROTSTATE] = *POC:normal active*

Equation 40 FR_MTSACRF[MTE] = 1

Equation 41

$$\text{CYCCNT} \,\&\, \text{FR_MTSACFR[CYCCNTMSK]} = \text{FR_MTSACFR[CYCCNTVAL]} \,\&\, \text{FR_MTSACFR[CYCCNTMSK]}$$

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if *Equation 39*, *Equation 42*, and *Equation 43* are fulfilled.

Equation 42 FR_MTSBCRF[MTE] = 1

Equation 43 CYCCNT & FR_MTSBCFR[CYCCNTMSK] = FR_MTSBCFR[CYCCNTVAL] & FR_MTSBCFR[CYCCNTMSK]

29.6.14 Key Slot Transmission

29.6.14.1 Key Slot Assignment

A key slot is assigned to the CC if the key_slot_id field in the *Protocol Configuration Register 18 (FR_PCR18)* is configured with a value greater than 0 and less or equal to number_of_static_slots in *Protocol Configuration Register 2 (FR_PCR2)*, otherwise no key slot is assigned.

29.6.14.2 Key Slot Transmission in *POC:startup*

If a key slot is assigned and the CC is in the *POC:startup* state, startup null frames will be transmitted as specified by *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

29.6.14.3 Key Slot Transmission in *POC:normal active*

If a key slot is assigned and the CC is in *POC:normal active*, a frame of the type as shown in *Table 481* is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see *Section 29.6.6.2.5: Message Transmission*). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see *Section 29.6.6.2.6: Null Frame Transmission*).

Table 481. Key Slot Frame Type

FR_PCR11[key_slot_used_for_sync]	FR_PCR11[key_slot_used_for_startup]	key slot frame type
0	0	normal frame
0	1	normal frame(1)
1	0	sync frame
1	1	startup frame

1. The frame transmitted has an semantically incorrect header and will be detected as an invalid frame at the receiver.

29.6.15 Sync Frame Filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, i.e. the SFFE control bit in the *Module Configuration Register (FR_MCR)* is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

29.6.15.1 Sync Frame Acceptance Filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the *Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)* and the mask is configured in the *Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)*. A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if *Equation 44* or *Equation 45*evaluates to true.

Equation 44 FR_MCR[SFFE] = 0

Equation 45

FID & FR_SFIDAFMR[FMSK] = FR_SFIDAFVR[FVAL] & FR_SFIDAFMR[FMSK]

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

29.6.15.2 Sync Frame Rejection Filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the *Sync Frame ID Rejection Filter Register (FR_SFIDRFR)*. A received synchronization frame with the frame ID FID passes the sync frame rejection filter if *Equation 46* or *Equation 47* evaluates to true.

Equation 46 FR_MCR[SFFE] = 0

Equation 47 FID ≠ FR_SFIDRFR[SYNFRID]

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

29.6.16 Strobe Signal Support

The CC provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in *Table 370*.

29.6.16.1 Strobe Signal Assignment

Each of the strobe signals listed in *Table 370* can be assigned to one of the four strobe ports using the *Strobe Signal Control Register (FR_STBSCR)*. To assign multiple strobe signals, the application must write multiple times to the *Strobe Signal Control Register (FR_STBSCR)* with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence.

1. Write to FR_STBSCR with WMD = 1 and SEL = N. (updates SEL field only)
2. Read STBCSR.
   The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

29.6.16.2 Strobe Signal Timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on

the FlexRay Bus. These signals are listed in *Table 370* with a negative clock offset. An example waveform is given in *Figure 562*.

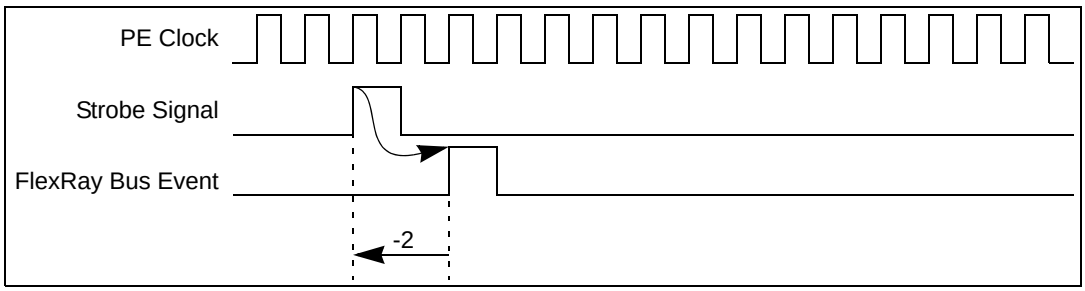


Figure 562. Strobe Signal Timing (type = pulse, clk_offset = -2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in *Table 370* with a positive clock offset. An example waveform is given in *Figure 563*.

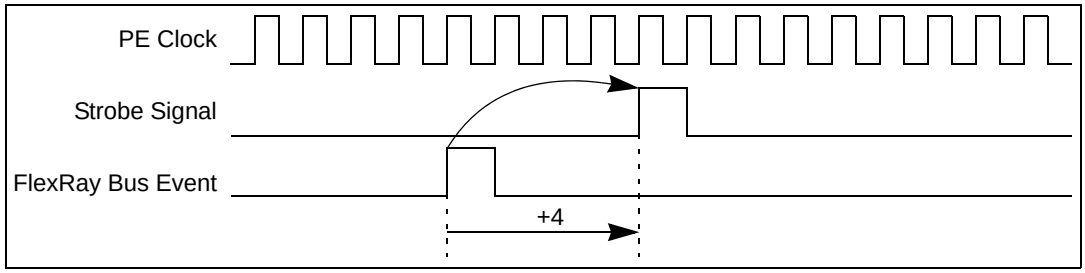


Figure 563. Strobe Signal Timing (type = pulse, clk_offset = +4)

29.6.17 Timer Support

The CC provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

29.6.17.1 Absolute Timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* is set at the macrotick start event, if *Equation 48* and *Equation 49* are fulfilled

Equation 48

$$CYCTR[CTCCNT] \,\&\, FR_TI1CYSR[T1_CYC_MSK] =$$
$$= FR_TI1CYSR[T1_CYC_VAL] \,\&\, FR_TI1CYSR[T1_CYC_MSK]$$

Equation 49 FR_MTCTR[MTCT] = FR_TI1MTOR[T1_MTOFFSET]

If the timer 1 interrupt enable bit TI1_IE in the *Protocol Interrupt Enable Register 0 (FR_PIER0)* is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

29.6.17.2 Absolute / Relative Timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the *Timer Configuration and Control Register (FR_TICCR)*. The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

29.6.17.2.1 Absolute Timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from *Timer 2 Configuration Register 0 (FR_TI2CR0)* and *Timer 2 Configuration Register 1 (FR_TI2CR1)* is used. On expiration of timer T2, the interrupt flag TI2_IF in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* is set. If the timer 1 interrupt enable bit TI1_IE in the *Protocol Interrupt Enable Register 0 (FR_PIER0)* is asserted, an interrupt request is generated.

29.6.17.2.2 Relative Timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the *Protocol Interrupt Flag Register 0 (FR_PIFR0)* is set, when the programmed amount of macroticks MT[31:0], defined by *Timer 2 Configuration Register 0 (FR_TI2CR0)* and *Timer 2 Configuration Register 1 (FR_TI2CR1)*, has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

29.6.18 Slot Status Monitoring

The CC provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is described in *Table 482*. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in *Figure 564*.

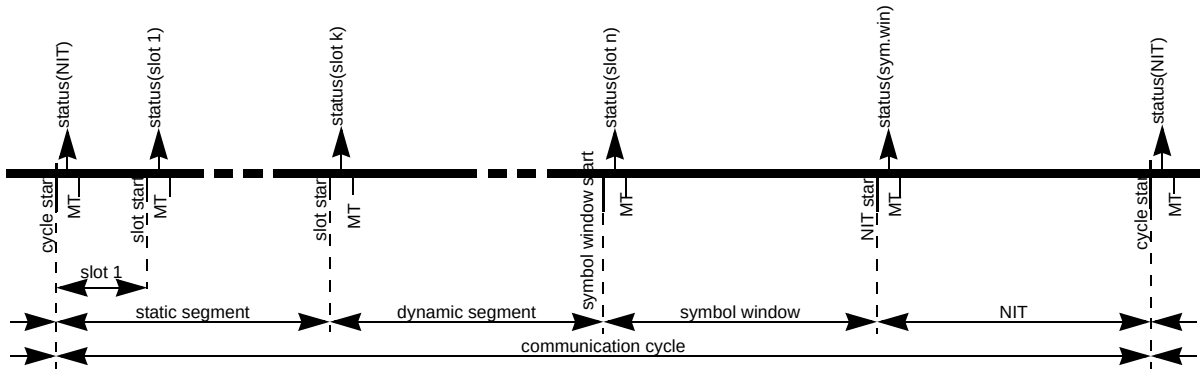


Figure 564. Slot Status Vector Update

Note: The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 482. Slot Status Content

	Status Content

static / dynamic Slot	slot related status
	vSS!ValidFrame - valid frame received
	vSS!SyntaxError - syntax error occurred while receiving
	vSS!ContentError - content error occurred while receiving
	vSS!BViolation - boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict - reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict - reception ongoing while transmission starts
	first valid - channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if vSS!ValidFrame = 1
	b) last received header, if vSS!ValidFrame = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator - Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator - Startup Frame Indicator
	vRF!Header!SyFIndicator - Sync Frame Indicator

Symbol Window	window related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* - valid Media Test Access Symbol received received frame related status see static/dynamic slot

Table 482. Slot Status Content

	Status Content
NIT	NIT related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 received frame related status see static/dynamic slot

29.6.18.1 Channel Status Error Counter Registers

The two channel status error counter registers, *Channel A Status Error Counter Register (FR_CASERCR)* and *Channel B Status Error Counter Register (FR_CBSERCR)*, incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

29.6.18.2 Protocol Status Registers

The *Protocol Status Register 2 (FR_PSR2)* provides slot status information about the Network Idle Time NIT and the Symbol Window. The *Protocol Status Register 3 (FR_PSR3)* provides aggregated slot status information.

29.6.18.3 Slot Status Registers

The eight slot status registers, *Slot Status Registers (FR_SSR0–FR_SSR7)*, can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in *Table 482*, except of the *first valid* indicator for non-transmission slots.

29.6.18.4 Slot Status Counter Registers

The CC provides four slot status error counter registers, *Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)*. Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the *Slot Status Counter Condition Register (FR_SSCCR)*, matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in *Figure 565*.

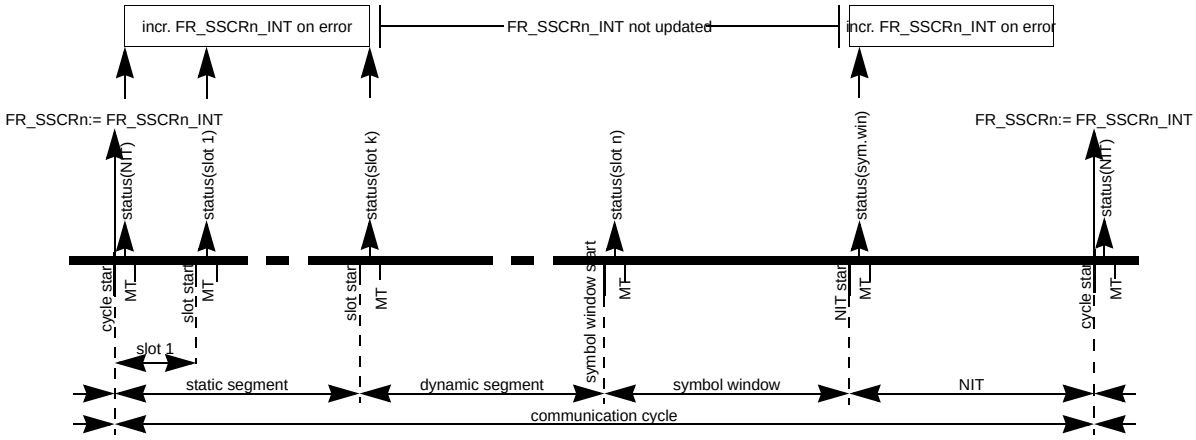


Figure 565. Slot Status Counting and FR_SSCRn Update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the FR_SSCRn register reflects, in cycle n, the status of the NIT of cycle n-2, and the status of all static slots and the symbol window of cycle n-1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter FR_SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (FR_SSCCRn[VFR] | FR_SSCCRn[SYF] | FR_SSCCRn[NUF] | FR_SSCCRn[SUF]) // count on frame condition
  = 1;

and

- ((~FR_SSCCRn[VFR] | *vSS!ValidFrame*) & // valid frame restriction
  (~FR_SSCCRn[SYF] | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
  (~FR_SSCCRn[NUF] | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
  (~FR_SSCCRn[SUF] | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction
  = 1;

Note: The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. slot related condition:

- ((FR_SSCCRn[STATUSMASK[3]] & *vSS!ContentError*) | // increment on content error
  (FR_SSCCRn[STATUSMASK[2]] & *vSS!SyntaxError*) | // increment on syntax error
  (FR_SSCCRn[STATUSMASK[1]] & *vSS!BViolation*) | // increment on boundary violation
  (FR_SSCCRn[STATUSMASK[0]] & *vSS!TxConflict*)) // increment on transmission conflict
  = 1;

If the slot status counter is in single cycle mode, i.e. FR_SSCCRn[MCY] = 0, the internal slot status counter FR_SSCRn_INT is reset at each cycle start. If the slot status counter is in the

multicycle mode, i.e. FR_SSCCRn[MCY] = 1, the counter is not reset and incremented, until the maximum value is reached.

29.6.18.5 Message Buffer Slot Status Field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in *Table 482* for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in *Section 29.6.6: Individual Message Buffer Functional Description*.

29.6.19 System Bus Access

This section provides a description of the system bus accesses failures and the related CC behavior. System bus access failures may occure when the CC transfers data to or from the flexray memory area.

The system bus access failure types are described in *Section 29.6.19.1: System Bus Access Failure Types*.

The behavior of the CC after the occurrence of a system bus access failure is described in *Section 29.6.19.2: System Bus Access Failure Response*.

29.6.19.1 System Bus Access Failure Types

This section describes the two types of system bus access failures.

29.6.19.1.1 System Bus Illegal Address Access

A system bus illegal address access is detected when the CC has used an illegal or invalid address to access the flexray system memory area. There are three conditions which are treated as a system bus illegal address access:

- The system bus subsystem detects an CC access to an illegal system memory address.
- The CC detects the usage of an data field offset with the value of 0.
- The CC detects a memory error while reading a data field offset from the CHI LRAM memory (see *Section 29.6.24.3.1: CHI LRAM Error Response after CC Read*).

If a system bus illegal address access is detected, the CC sets the ILSA_EF flag in the *CHI Error Flag Register (FR_CHIERFR)*.

29.6.19.1.2 System Bus Access Timeout

A system bus access timout is detected if an access to the flexray memory area is not finished in time. The timeout value is derived from the SYMATOR[TIMEOUT] setting (see *Section 29.7.1.1: Configure System Memory Access Time-Out Register (FR_SYMATOR)*"

If a system bus access timout is detected, the CC sets the SBCF_EF flag in the *CHI Error Flag Register (FR_CHIERFR)*.

29.6.19.2 System Bus Access Failure Response

This section describes the two types of behavior of the CC after the occurrence of a system bus access failure. The actual behavior is defined by the SBFF bit in the *Module Configuration Register (FR_MCR)*.

29.6.19.2.1 Continue after System Bus Access Failure

If the SBFF bit in the *Module Configuration Register (FR_MCR)* is 0, the CC will continue its operation after the occurrence of the system bus access failure, but will not generate any system bus accesses until the start of the next communication cycle.Since no data are read from or written to the flexray memory area, no messages are received or transmitted. Consequently, none of the individual message buffers or receive FIFOs will be updated until the next communication cycle starts.

If a frame is under transmission when the system bus failure occurs, a correct frame is generated with the remaining header and frame data are replaced by all zeros. Depending on the point in time this can affect the PPI bit, the Header CRC, the Payload Length in case of an dynamic slot, and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of next communication cycle.

29.6.19.2.2 Freeze after System Bus Access Failure

If the SBFF bit in the *Module Configuration Register (FR_MCR)* is set to 1, the CC will go into the freeze mode immediately after the occurrence of one of the system bus access failures.

29.6.20 Interrupt Support

The CC provides 172 individual interrupt sources and five combined interrupt sources.

29.6.20.1 Individual Interrupt Sources

29.6.20.1.1 Message Buffer Interrupts

The CC provides 128 message buffer interrupt sources.

Each individual message buffer provides an interrupt flag FR_MBCCSRn[MBIF] and an interrupt enable bit FR_MBCCSRn[MBIE]. The CC sets the interrupt flag when the slot status of the message buffer was updated. If the interrupt enable bit is asserted, an interrupt request is generated.

29.6.20.1.2 FIFO Interrupts

The CC provides 2 FIFO interrupt sources.

Each of the 2 FIFO provides a Receive FIFO Almost Full Interrupt Flag. The CC sets the Receive FIFO Almost Full Interrupt Flags (FR_GIFER[FAFBIF], FR_GIFER[FAFAIF]) in the *Global Interrupt Flag and Enable Register (FR_GIFER)* if the corresponding Receive FIFO fill level exceeds the defined watermark.

29.6.20.1.3 Wakeup Interrupt

The CC provides one interrupt source related to the wakeup.

The CC sets the Wakeup Interrupt Flag FR_GIFER[WUPIF] when it has received a wakeup symbol on the FlexRay bus. The CC generates an interrupt request if the interrupt enable bit FR_GIFER[WUPIE] is asserted.

29.6.20.1.4 Protocol Interrupts

The CC provides 25 interrupt sources for protocol related events. For details, see *Protocol Interrupt Flag Register 0 (FR_PIFR0)* and *Protocol Interrupt Flag Register 1 (FR_PIFR1)*. Each interrupt source has its own interrupt enable bit.

29.6.20.1.5 CHI Interrupts

The CC provides 16 interrupt sources for CHI related error events. For details, see *CHI Error Flag Register (FR_CHIERFR)*. There is one common interrupt enable bit FR_GIFER[CHIE] for all CHI error interrupt sources.

29.6.20.2 Combined Interrupt Sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

29.6.20.2.1 Receive Message Buffer Interrupt

The Receive Message Buffer Interrupt request is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[RBIE] is set.

29.6.20.2.2 Transmit Message Buffer Interrupt

The Transmit Message Buffer Interrupt request is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[TBIE] is asserted.

29.6.20.2.3 Protocol Interrupt

The Protocol Interrupt request is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[PRIE] is set.

29.6.20.2.4 CHI Interrupt

The CHI Interrupt request is generated when at least one of the individual chi error interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[CHIE] is set.

29.6.20.2.5 Module Interrupt

The Module Interrupt request is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[MIE] is set.

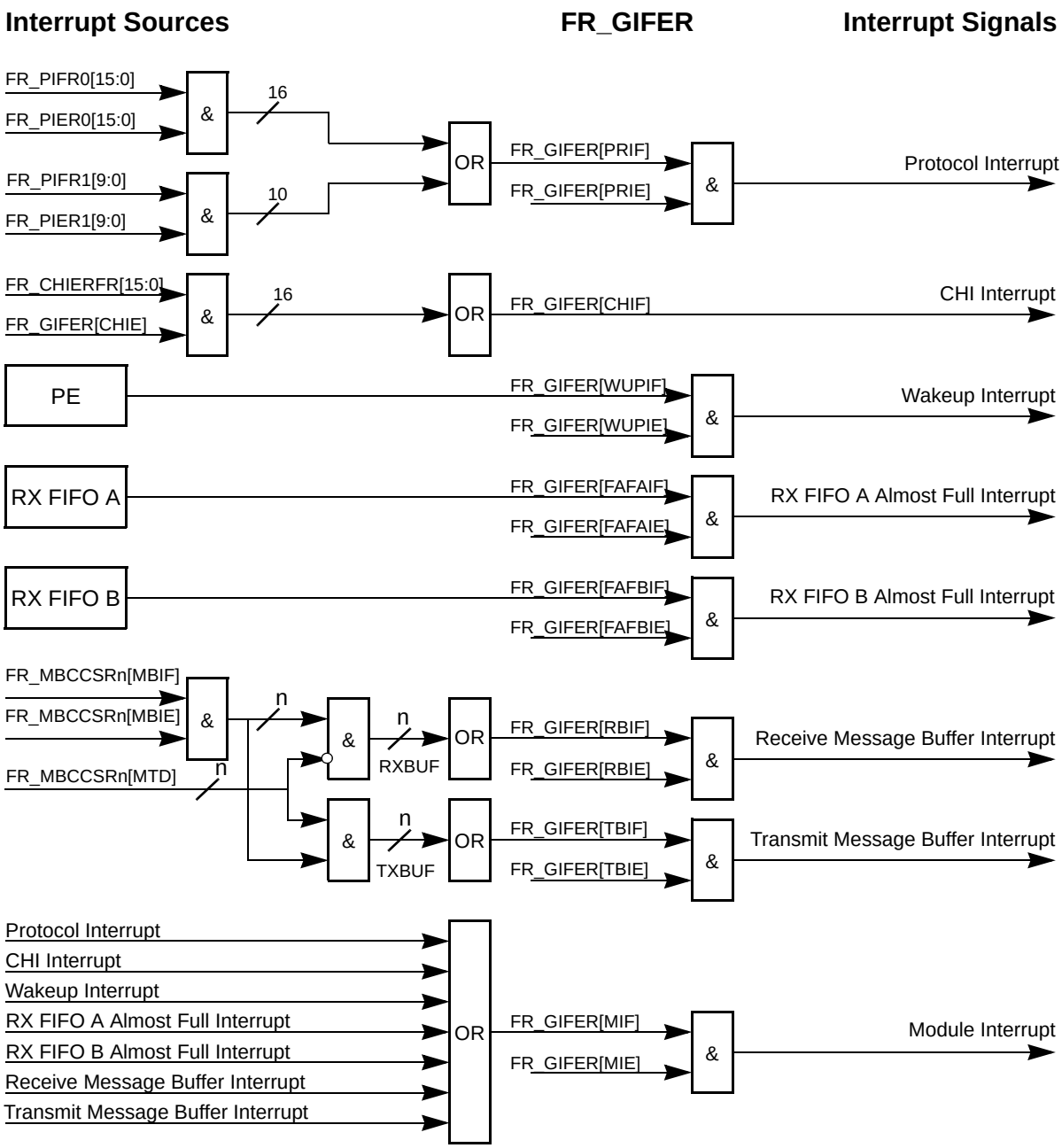


Figure 566. Scheme of FR_GIFER interrupt signal generation

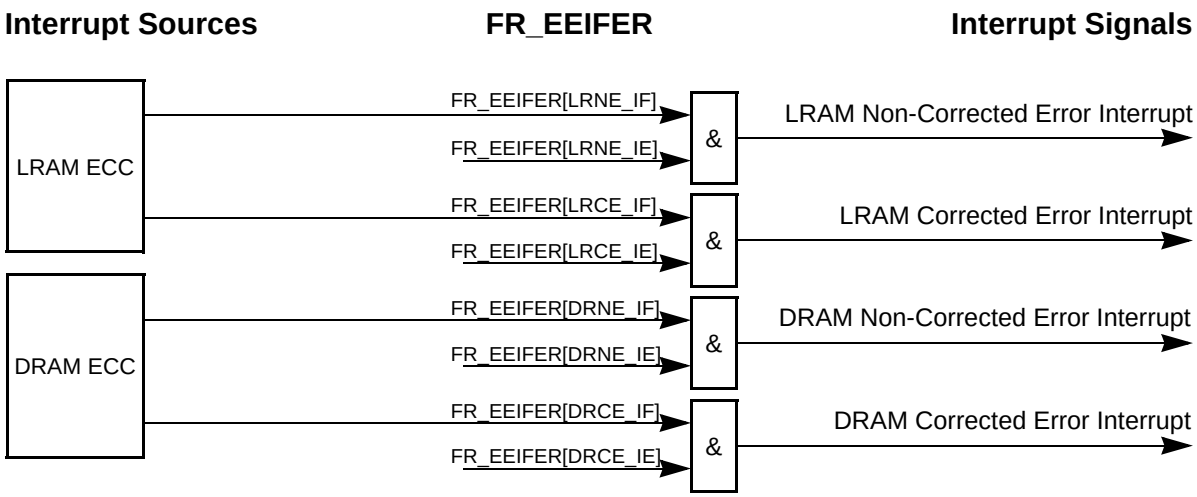


Figure 567. Scheme of FR_EEIFER interrupt signal generation

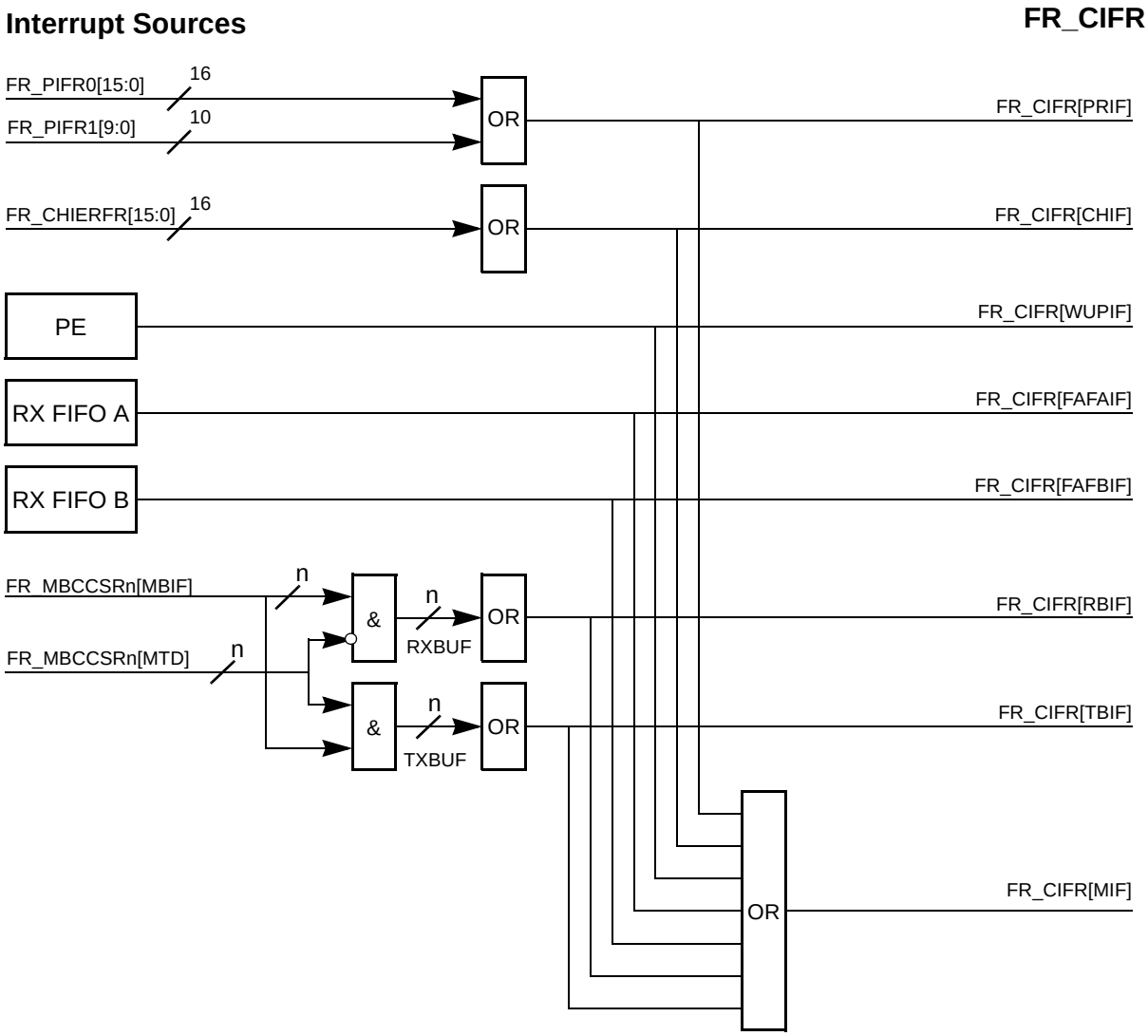


Figure 568. Scheme of FR_CIFR flags generation

29.6.21 Lower Bit Rate Support

The CC supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset.* The application configures the FlexRay channel bit rate by setting the BITRATE field in the Module Configuration Register (FR_MCR). The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in *Table 483*.

Table 483. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	FR_MCR[BITRATE]	*pdMicrotick* [ns]	*gdSampleClockPeriod* [ns]	pSamplesPerMicrotick	cSamplesPerBit	cStrobeOffset
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

Note: The bit rate of 8 Mbit/s is not defined by the FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

29.6.22 PE Data Memory (PE DRAM)

The PE Data Memory (PE DRAM) is 128 word, 16-bit wide memory with byte access, which contains the program data of the PE internal CPU. The PE DRAM is divided into two banks, 8-bit each. The memory data [7:0] are assigned to BANK0, the memory data [15:8] are assigned to BANK1.

Table 484. PE DRAM Layout

ADDR	BANK1	BANK0
0x00	byte1	byte0
0x01	byte3	byte2
...		
0x7F	byte255	byte254

The FlexRay module provides means to access the PE DRAM from the application. The PE DRAM application access is initiated and controlled via *PE DRAM Access Register (FR_PEDRAR)* and *PE DRAM Data Register (FR_PEDRDR)*. This functionality is used to check the memory error detection.

29.6.22.1 PE DRAM Read Access

A read access from the PE DRAM can be initiated in any protocol state. The following sequence describes a read access from the PE DRAM address 0x70.

1. FR_PEDRAR:= 0x50E0;
   // INST=0x5; ADDR=070
2. wait until FR_PEDRAR[DAD] == 1;
   // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA];
   // read PE DRAM data

The read access is handled by the PE internal CPU with the lowest execution priority. This may cause an response delay with a maximum of 1000 PE clock cycle (25us).

29.6.22.2 PE DRAM Write Access

A write access into the PE DRAM can be initiated in any protocol state. The following sequence describes a write access to the PE DRAM address 0x70.

1. FR_PEDRDR:= DATA;
   // write value to be written into data register
2. FR_PEDRAR:= 0x30E0;
   // INST=0x3; ADDR=0x70
3. wait until FR_PEDRAR[DAD] == 1;
   // wait for end of PE DRAM access
4. val = FR_PEDRDR[DATA];
   // read back PE DRAM data

The write access is handled by the PE internal CPU with the lowest execution priority. This may causes an response delay with a maximum of 1000 PE clock cycle (25us).

If the conditions given in *Section 29.6.22.3: PE DRAM Write Access Limitations* are fulfilled, the data provided in *PE DRAM Data Register (FR_PEDRDR)* are written into the PE DRAM, read back in the next clock cycle and stored into the *PE DRAM Data Register (FR_PEDRDR)*. Otherwise, data are not written into the PE DRAM and 0x0000 is stored into the *PE DRAM Data Register (FR_PEDRDR)*.

29.6.22.3 PE DRAM Write Access Limitations

The PE DRAM is used by the protocol engine if the module is not in *POC:default config* state. The only address not used by the protocol engine is 0x70. To prevent the corruption of protocol engine data the following PE DRAM write access limitations apply for application writes.

1. When the module is in *POC:default config* state, all PE DRAM addresses are writable.
2. When the module is not in *POC:default config* state, only PE DRAM address 0x70 is writable.

29.6.23 CHI Lookup-Table Memory (CHI LRAM)

The CHI Lookup-Table Memory (CHI LRAM) is an CHI internal memory which contains the message buffer configuration data and the data field offsets for the physical message buffers. The configuration data for two message buffers or 6 data field offsets are contained in one memory row. The CHI LRAM is divided into 6 memory BANKs.

Table 485. CHI LRAM Layout

ADR	BANK5	BANK4	BANK3	BANK2	BANK1	BANK0
0x00	FR_MBIDXR1	FR_MBFIDR1	FR_MBCCFR1	FR_MBIDXR0	FR_MBFIDR0	FR_MBCCFR0
0x01	FR_MBIDXR3	FR_MBFIDR3	FR_MBCCFR3	FR_MBIDXR2	FR_MBFIDR2	FR_MBCCFR2
...						
0x3F	FR_MBIDXR127	FR_MBFIDR127	FR_MBCCFR127	FR_MBIDXR126	FR_MBFIDR126	FR_MBCCFR126
0x40	FR_MBDOR5	FR_MBDOR4	FR_MBDOR3	FR_MBDOR2	FR_MBDOR1	FR_MBDOR0
...						
0x55	FR_MBDOR131	FR_MBDOR130	FR_MBDOR129	FR_MBDOR128	FR_MBDOR127	FR_MBDOR126
0x56	FR_LEETR5	FR_LEETR4	FR_LEETR3	FR_LEETR2	FR_LEETR1	FR_LEETR0

29.6.23.1 CHI LRAM Read and Write Access

The CHI LRAM is accessed by the application via regular register read and write accesses.

29.6.24 Memory Content Error Detection

The FlexRay module provides integrated memory content error detection for both the CHI LRAM and PE DRAM, and memory content error correction for the PE DRAM. The memory error detection for the CHI LRAM uses an standard Hamming code with a Hamming distance of 3 and detects all single-bit and double-bit errors (SEDDED). The memory error detection and correction for the PE DRAM uses an enhanced Hamming code with a Hamming distance of 4 and detects and corrects all single-bit errors and detects all double-bit errors (SECDED).

This section describes the reporting of the occurrence of memory content errors, the reaction of the module on the occurrence, and how the application can inject memory errors in order to trigger the report and response behavior.

29.6.24.1 Memory Error Types

A memory error is the distortion of one or more bits read out of the memory. The reading of the values of all zeros and all ones is considered as an special case. The FlexRay module detects and indicates the memory errors as shown in *Table 486*. The entries on the top have higher priority.

Each memory read access reads out *all* banks of the addressed row, and runs error detection on *all* banks, even in the case that the application has triggered a read from only one bank. This may lead to the reporting of an memory error if at least one bank contains a memory error, even if an error free bank has been read.

Table 486. Detected Memory Error Types

Memory	Priority	Memory Data	Indication
CHI LRAM	0 (highest)	All Zero's	No Error - Valid Data
PE DRAM			Non-Corrected Error
CHI LRAM		All One's	Non-Corrected Error
PE DRAM			
CHI LRAM	1 (lowest)	One Bit Flipped	Non-Corrected Error
PE DRAM			Corrected Error
CHI LRAM		Two Bits Flipped	Non-Corrected Error
PE DRAM			
CHI LRAM		Three or more Bits Flipped	one out of {No error, Non-Corrected Error}, defined by coding given in *Section 29.6.24.2.3: CHI LRAM Checkbits* and *Section 29.6.24.2.3: CHI LRAM Checkbits*
PE DRAM			one out of {No error, Corrected Error, Non-Corrected Error}, defined by coding given in *Section 29.6.24.2.1: PE DRAM Checkbits* and *Section 29.6.24.2.2: PE DRAM Syndrome*

29.6.24.2 Memory Error Reporting

The memory error reporting is enabled only if the ECC functionality enable bit ECCE in the *Module Configuration Register (FR_MCR)* is set.

For each of the two memories exists two sets of internal registers to store the detection of one corrected and one non-corrected memory error.

If a memory error is detected, the module checks whether the related error interrupt flag in the *ECC Error Interrupt Flag and Enable Register (FR_EEIFER)* is set.

- If the error interrupt flag is set, the related internal error reporting register is not updated and the related error overflow flag is set to 1 to indicate a loss of error condition.
- If the error interrupt flag is not set, the internal reporting register is updated and the error interrupt flag is set to 1. If two or more memory errors of the same type are detected, the error for the bank with the lower bank number will be reported, and the error overflow flag will be set to 1.

If a memory error is detected for at least two banks of one memory, the related error overflow flag is set to 1 to indicate a loss of error condition.

29.6.24.2.1 PE DRAM Checkbits

The coding of the checkbits reported in *ECC Error Report Code Register (FR_EERCR)* for PE DRAM memory errors is shown in *Table 488*. This table shows the implemented enhanced Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 487. PE DRAM checkbits coding

CODE	CODE				DATA							
	3	2	1	0	7	6	5	4	3	2	1	0
4(1)	X	X	X	X	X	X	X	X	X	X	X	X
3(2)	-	-	-	-	X	X	X	X	-	-	-	-
2	-	-	-	-	X	-	-	-	X	X	X	-
1	-	-	-	-	-	X	X	-	X	X	-	X
0	-	-	-	-	-	X	-	X	X	-	X	X

1. The checkbit CODE[4] is set to 1 if and only if there is a even number of 1's in columns with X.
2. The checkbits CODE[3]... CODE[0] are set to 1 if and only if there is a odd number of 1's in all columns with X.

This coding of the checkbit ensures that neither 0x000 nor 0xFFF are valid code words and written into the memory.

29.6.24.2.2 PE DRAM Syndrome

The coding of the syndrome reported in *ECC Error Report Code Register (FR_EERCR)* for PE DRAM memory errors is shown in *Table 488*.

Table 488. FR_EERCR[CODE] PE DRAM Syndrome Coding

FR_EERCR[CODE]		Description
[4]	[3:0]	
0x1	0x0	No Error (Never appears in error report registers)
0x0	0x0	If data == 0: Non Corrected Error (Dedicated Handling of All Zero Code Word) If data!= 0: Corrected Error (Parity Bit 4)
0x0	0x1	Corrected Error (Parity Bit 0)
0x0	0x2	Corrected Error (Parity Bit 1)
0x0	0x3	Corrected Error (Data Bit 0)
0x0	0x4	Corrected Error (Parity Bit 2)
0x0	0x5	Corrected Error (Data Bit 1)
0x0	0x6	Corrected Error (Data Bit 2)
0x0	0x7	Corrected Error (Data Bit 3)
0x0	0x8	Corrected Error (Parity Bit 3)
0x0	0x9	Corrected Error (Data Bit 4)
0x0	0xA	Corrected Error (Data Bit 5)
0x0	0xB	Corrected Error (Data Bit 6)
0x0	0xC	Corrected Error (Data Bit 7)
0x0	0xD-0xF	Non-Corrected Error
0x1	0x1-0xF	Non-Corrected Error

29.6.24.2.3 CHI LRAM Checkbits

The coding of the checkbits reported in *ECC Error Report Code Register (FR_EERCR)* for CHI LRAM memory errors is shown in *Table 489*. This table shows the implemented Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 489. CHI LRAM checkbits coding

CODE(1)	DATA															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	X	X	X	X	X	X	X	-	-	-	-
2	X	X	-	-	-	X	X	X	X	-	-	-	X	X	X	-
1	-	-	X	X	-	X	X	-	-	X	X	-	X	X	-	X
0	X	-	X	-	X	X	-	X	-	X	-	X	X	-	X	X

1. The checkbit CODE[n] is set to 1 if and only if there is a odd number of 1's in all columns with X.

29.6.24.2.4 CHI LRAM Syndrome

The coding of the syndrome reported in *ECC Error Report Code Register (FR_EERCR)* for CHI LRAM memory errors is shown in *Table 490*.

Table 490. FR_EERCR[CODE] CHI LRAM Syndrome Coding

FR_EERCR[CODE]	Description
0x00	No Error (Never appears in error report registers)
0x01-0x1F	Non Corrected Error

29.6.24.3 Memory Error Response

The memory error response is enabled only when the ECC functionality enable bit ECCE in the *Module Configuration Register (FR_MCR)* is set.

In case of the detection of a *corrected* memory error, the FlexRay module continues its normal operation using the corrected data word. This section describes the behavior of the FlexRay module after the detection of a *non-corrected* memory error.

29.6.24.3.1 CHI LRAM Error Response after CC Read

When the CC is out of the *POC:default config* state, it reads the configuration data and the data field offsets of all utilized message buffers in every slot and in the NIT. If a non-corrected memory error is detected during this module read access the error response of the module depends from LRAM location where the error occurred.

- If the LRAM address belongs to physical message buffer configuration data the FlexRay module will consider the affected message buffer as disabled for the current search and will exclude this buffer from the search. The configuration of the affected message buffer is not changed.

If the affected message buffer is a tx message buffer, no frame will be transmitted from this message buffer in the next slot. If the affected message buffer is a rx message buffer, no frame will be received to this message buffer in the next slot.

- If the LRAM address belongs to the data field offset area and the related physical message buffer is used for Rx or Tx the first access to the system memory caused by payload read or write yields to the assertion of the FR_CHIERFR[ILSA_EF]. No memory access occurs w.r.t. payload access is performed for the complete frame.

29.6.24.3.2 CHI LRAM Error Response after Application Read

The application can read the content of the CHI LRAM via reading the FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn, FR_MBDORn, and FR_LEETRn registers. If a non-corrected memory error is detected during this kind of read access, the module indicates the detected memory error, delivers the non-corrected data read and continues its normal operation.

29.6.24.3.3 PE DRAM Error Response after CC Read

If the CC detects an non-corrected memory error during internal read of program data which is contained in PE DRAM, this is considered as an fatal protocol error and the module enters the protocol freeze state immediately.

29.6.24.3.4 PE DRAM Error Response after Application Read in *POC:default config* state

If the CC detects an non-corrected memory error during an application triggered read from any PE DRAM address and the protocol is in the *POC:default config* state, this is considered as an fatal protocol error and the module enters the protocol freeze state. This behavior allows for checking the freeze functionality in case of the detection of non-corrected errors.

29.6.24.3.5 PE DRAM Error Response after Application Read out of *POC:default config*

If the CC detects an non-corrected memory error during an application triggered read from any PE DRAM address, and the protocol is not in the *POC:default config* state, this error is not considered as an fatal error and the protocol state is not changed. This prevents any interference of the running protocol by PE DRAM error injection reads.

29.6.25 Memory Error Injection

The error injection functionality is used by the application to inject data errors into the memories to trigger and check the memory error detection functionality.

The error injection is enabled only if the ECC functionality enable bit ECCE in the *Module Configuration Register (FR_MCR)* and the error injection enable control bit EIE in the *ECC Error Report and Injection Control Register (FR_EERICR)* are set.

The error injection mode is configured by the EIM configuration bit in the *ECC Error Report and Injection Control Register (FR_EERICR)*.When the error injection is enabled, each write access to the configured memory location will be distorted.

The injector has the same behavior for FlexRay module memory writes and application memory writes.

29.6.25.1 CHI LRAM Error Injection

The following sequence describes an memory error injection sequence for the CHI LRAM memory. This sequence consists of the setup of the error injector followed by an application

triggered write access to provoke an distortion of the memory content. The content of the CHI LRAM is described in *Table 485*.

When the CC is in *POC:default config*, there are no limitations for the error injection and no impacts of error injection to the application. For error injection out of *POC:default config* see *Section 29.7.3: Memory Error Injection out of POC:default config*.

Injector Setup:

1. FR_MCR[ECCE]:= 1;
   // enable ecc functionality
2. FR_EERICE[EIE]:=I_MODE;
   // configure error injection mode
3. FR_EEIAR[MID]:= 1;
   // select CHI LRAM for error injection
4. FR_EEIAR[BANK]:= I_BANK;
   // select bank for error injection; I_BANK = {0,1,2,3,4,5}
5. FR_EEIAR[ADDR]:= I_ADDR;
   // select address for error injection; I_ADDR <= 0x56
6. FR_EEIDR[DATA]:= D_DIST;
   // define data distortion pattern
7. FR_EEICR[CODE]:= C_DIST;
   // define checkbit distortion pattern
8. FR_EERICE[EIE]:=1;
   // enable error injection

Application Write Access:

If (I_BANK==0) -> FR_MBCCFR(2n) / FR_MBDOR(6k) / FR_LEETR0 := DATA;
If (I_BANK==1) -> FR_MBFIDR(2n) / FR_MBDOR(6k+1) / FR_LEETR1 := DATA;
If (I_BANK==2) -> FR_MBIDXR(2n) / FR_MBDOR(6k+2) / FR_LEETR2 := DATA;
If (I_BANK==3) -> FR_MBCCFR(2n+1) / FR_MBDOR(6k+3) / FR_LEETR3 := DATA;
If (I_BANK==4) -> FR_MBFIDR(2n+1) / FR_MBDOR(6k+4) / FR_LEETR4 := DATA;
If (I_BANK==5) -> FR_MBIDXR(2n+1) / FR_MBDOR(6k+5) / FR_LEETR5 := DATA;
// write DATA to the defined injection bank and injection address (see *Table 485*).

29.6.25.2 PE DRAM Error Injection

The following sequence describes an memory error injection sequence for the PE DRAM memory. This sequence consists of the setup of the error injector followed by an application triggered write access to provoke an distortion of the memory content.

When the FlexRay module is in *POC:default config*, there are no limitations for the error injection and no impacts of error injection to the application. For error injection out of *POC:default config* see *Section 29.7.3.2: PE DRAM Error Injection out of POC:default config*.

Injector Setup:

1. FR_MCR[ECCE]:= 1;
   // enable ecc functionality
2. FR_EERICE[EIE]:=I_MODE;
   // configure error injection mode
3. FR_EEIAR[MID]:= 0;
   // select PE DRAM for error injection
4. FR_EEIAR[BANK]:= I_BANK;
   // define bank for error injection; I_BANK = {0,1}
5. FR_EEIAR[ADDR]:= I_ADDR;
   // define address for error injection; I_ADDR <= 0x7F
6. FR_EEIDR[DATA]:= D_DIST;
   // define data distortion pattern
7. FR_EEICR[CODE]:= C_DIST;
   // define checkbit distortion pattern
8. FR_EERICE[EIE]:=1;
   // enable error injection

Application Write Access (e.g. I_ADDR=0x70):

1. FR_PEDRAR:= 0x30E0;
   // INST=0x3; ADDR=0x70
2. wait until FR_PEDRAR[DAD] == 1;
   // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; |
   // get read back PE DRAM data

Note: The write access to the PE DRAM triggers an subsequent read access from PE DRAM in the next cycle, which triggers the detection of the distorted data.

29.7 Application Information

29.7.1 Module Configuration

This section describes essential parts of the module configuration.

29.7.1.1 Configure System Memory Access Time-Out Register (FR_SYMATOR)

To ensure reliable operation of the CC, the application must ensure that the TIMEOUT value in *System Memory Access Time-Out Register (FR_SYMATOR)* and the CHI clock frequency f_{CHI} in MHz fulfill *Equation 50*[o].

Equation 50 $$0 \leq \text{SYMATOR[TIMEOUT]} \leq \lfloor 0.45 \cdot f_{CHI} - 8 \rfloor$$

For a given SYMATOR[TIMEOUT] value, f_{CHI} can be increased without causing unreliable operation of the CC. The same holds for reducing the SYMATOR[TIMEOUT] value for a given f_{CHI}.

Some examples for maximum values of the SYMATOR[TIMEOUT] for a minimum CHI frequency are given in *Table 491*.

o. see *Section 29.3: Controller Host Interface Clocking* for all constraints of minimum CHI clock frequency.

Table 491. Maximum SYMATOR[TIMEOUT] examples

f_{CHI}	SYMATOR[TIMEOUT]	f_{CHI}	SYMATOR[TIMEOUT]
>= 18 MHz	0	>= 100 MHz	<= 37
>= 23 MHz	<= 2	>= 120 MHz	<= 46
>= 27 MHz	<= 4	>= 140 MHz	<= 55
>= 32 MHz	<= 6	>= 160 MHz	<= 64
>= 60 MHz	<= 19	>= 180 MHz	<= 73
>= 80 MHz	<= 28	>= 200 MHz	<= 82

29.7.1.1.1 System Bus Wait State Constraints

The SYMATOR[TIMEOUT] value corresponds directly to a certain acceptable number of wait states on the system bus.

For single channel configurations and if the sync frame table generation functionality is *not* used (FR_SFTCCSR[SDVEN,SIDEN] = 0) no timeout will be detected if less than 2*SYMATOR[TIMEOUT]+1 wait states will be seen on the system bus for each system bus access.

For dual channel configurations, or if the sync frame table generation functionality is used, no timeout will be detected if less than SYMATOR[TIMEOUT]-1 wait states will be seen on the system bus for each system bus access.

29.7.1.2 Configure Data Field Offsets

The data field offsets are located in the *Message Buffer Data Field Offset Registers (FR_MBDORn)* and *Receive FIFO Start Data Offset Register (FR_RFSDOR)*. The application has to configure the data field offset values for all message buffers which are used.

The reset value of all data field offsets FR_MBDORn[MBDO] and FR_RFSDOR[SDO] is 0. This value is considered to be illegal (see *Section 29.6.19.1.1: System Bus Illegal Address Access*).

29.7.2 Initialization Sequence

This section describes the required steps to initialize the CC. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the CC.

29.7.2.1 Module Initialization

This section describes the module related initialization steps after a system reset.

1. Configure CC.
   a) configure the control bits in the *Module Configuration Register (FR_MCR)*
   b) configure the system memory base address in *System Memory Base Address Register (FR_SYMBADR)*
2. Enable the CC.
   a) write 1 to the module enable bit MEN in the *Module Configuration Register (FR_MCR)*

The CC now enters the Normal Mode. The application can commence with the protocol initialization described in *Section 29.7.2.2: Protocol Initialization*.

29.7.2.2 Protocol Initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
   a) issue CONFIG command via *Protocol Operation Control Register (FR_POCR)*
   b) wait for *POC:config* in *Protocol Status Register 0 (FR_PSR0)*
   c) configure the FR_PCR0,..., FR_PCR30 registers to set all protocol parameters
2. Configure the Message Buffers and FIFOs.
   a) set the number of message buffers used and the message buffer segmentation in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*
   b) define the message buffer data size in the *Message Buffer Data Size Register (FR_MBDSR)*
   c) configure each message buffer by setting the configuration values in the *Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)*, *Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)*, *Message Buffer Frame ID Registers (FR_MBFIDRn)*, *Message Buffer Index Registers (FR_MBIDXRn)*
   d) configure the FIFOs
   e) issue CONFIG_COMPLETE command via *Protocol Operation Control Register (FR_POCR)*
   f) wait for *POC:ready* in *Protocol Status Register 0 (FR_PSR0)*

After this sequence, the CC is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

29.7.2.3 CHI LRAM Initialization

The initialization of the CHI LRAM is performed by the CC when it leaves the Disabled Mode. The unitization runs for 87 CHI clock cycles. All fields in the FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBDORn, and LEETRn registers are initialized to 0. All application read or write accesses to these registers are delayed until the initialization is finished.

29.7.2.4 PE DRAM Initialization

The PE DRAM initialization is performed by the CC in the *POC:default config* state. This initialization runs for 4.8 µs, and will delay the state transition from *POC:default config* into *POC:config*.

29.7.3 Memory Error Injection out of *POC:default config*

This section provides information for application driven memory error injection out if *POC:default config.* The CC provides means to inject memory errors from the application without any impacts to the internal protocol operation of the CC.

29.7.3.1 CHI LRAM Error Injection out of *POC:default config*

The CC will never perform any internal read access from the *LRAM ECC Error Test Registers (FR_LEETRn)*. Any memory errors injected into these CHI LRAM locations will never be detected by internal access, independent from the protocol state.

The application should use these registers and related CHI LRAM location to inject memory errors into the CHI LRAM. The injection sequence is described in *Section 29.6.25.1: CHI LRAM Error Injection*.

29.7.3.2 PE DRAM Error Injection out of *POC:default config*

The CC will never perform any internal read access from the PE DRAM address 0x70. This is the only one PE DRAM address writable by the application out of the *POC:default config* state.

The application should use these PE DRAM location to inject memory errors into the PE DRAM. The injection sequence is described in *Section 29.6.25.2: PE DRAM Error Injection*.

29.7.4 Shut Down Sequence

This section describes a secure shut down sequence to stop the CC gracefully. The main targets of this sequence are

- finish all ongoing reception and transmission
- do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a graceful shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) repeatedly write 1 to FR_MBCCSRn[EDT] until FR_MBCCSRn[EDS] == 0.
2. Stop Protocol Engine.
   a) issue HALT command via *Protocol Operation Control Register (FR_POCR)*
   b) wait for *POC:halt* in *Protocol Status Register 0 (FR_PSR0)*

29.7.5 Number of Usable Message Buffers

This section describes the required minimum CHI clock frequency for a specified number of utilized message buffers configured in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)*, a configured minislot length *gdMinislot*, and a configured nominal macrotick length *gdMacrotick*[p].

Additional constraints for the minimum CHI clock frequency are given in *Section 29.3: Controller Host Interface Clocking*.

The CC uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search is started at the start of slot and must be finished before the start of the next slot.

The shortest FlexRay slot is an corrected empty dynamic slot. An corrected empty dynamic slot is a minislot and consists of *gdMinislot* corrected macroticks with a duration of *gdMacrotick*. The minimum duration of an corrected macrotick is $gdMacrotick_{min}$ = 39 µµT. This results in a minimum length of an correct slot

Equation 51 $\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot$

The message buffer search engine runs on the CHI clock and evaluates one individual message buffer per CHI clock cycle. For internal status update operations and to account for

p. see *Section 29.3: Controller Host Interface Clocking* for all constraints of minimum CHI clock frequency.

clock domain crossing jitter, an additional amount of 27 CHI clock cycles is required to ensure correct search engine operation.

For a given number of utilized message buffers FR_MBSSUTR[LAST_MB_UTIL] + 1 and for a given CHI clock frequency f_{chi}, this results in a search duration of

Equation 52 $$\Delta_{search} = \frac{1}{f_{chi}} \cdot (FR_MBSSUTR[LAST_MB_UTIL]+27)$$

The message buffer search must be finished within one slot which requires that *Equation 53* must be fulfilled:

Equation 53 $$\Delta_{search} \leq \Delta_{slotmin}$$

This results in the formula given in *Equation 54* which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

Equation 54 $$f_{chi} \geq \frac{(FR_MBSSUTR[LAST_MB_UTIL]+27)}{39 \cdot pdMicrotick \cdot gdMinislot}$$

The required minimum CHI Clock frequency for a selected set of relevant protocol parameters and for the LAST_MB_UTIL field in the *Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)* set to 127 is given in *Table 492*.

Table 492. Minimum f_{chi} [MHz] examples (128 message buffers used)

pdMicrotick **[ns]**	**gdMinislot**					
	2	**3**	**4**	**5**	**6**	**7**
25.0	79.5	53	39.8	31.8	26.5	22.8
50.0	39.8	26.5	19.9	15.9	13.3	11.4

Note: If the minimum CHI frequency is not met the CHIERFR[MBS_EF] flag is set. Refer to Section 29.5.2.17: CHI Error Flag Register (FR_CHIERFR) for details.

29.7.6 Protocol Control Command Execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of *Table 374* by writing the command to the POCCMD field of the *Protocol Operation Control Register (FR_POCR)*. As a result the CC sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in *Table 493*. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the CC asserts the

illegal protocol command interrupt flag IPC_IF in the *Protocol Interrupt Flag Register 1 (FR_PIFR1)*. The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by *Table 493*. If the application issues the FREEZE or READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 493. Protocol Control Command Priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
FREEZE	(highest) 1	none	
READY	2		
CONFIG_COMPLETE	3		
ALL_SLOTS	4	FREEZE, READY, CONFIG_COMPLET,fatal protocol error	FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		
RUN	6		FREEZE, fatal protocol error
WAKEUP	7		FREEZE, fatal protocol error
DEFAULT_CONFIG	8		FREEZE, fatal protocol error
CONFIG	9		
HALT	(lowest) 10		FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

29.7.7 Message Buffer Search on Simple Message Buffer Configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The FIFO behavior is not considered in this section.

29.7.7.1 Simple Message Buffer Configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and has following configuration

Table 494. Transmit Buffer Configuration

Register	Field	Value	Description
FR_MBCCSR*t*	MTD	1	transmit buffer
FR_MBCCFR*t*	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = {4n} = {0,4,8,12,...}
	CCFVAL	000000	
FR_MBFIDR*t*	FID	S	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit FR_MBCCSRt[CMT] and the lock bit FR_MBCCSRt[LCKS].

The receive message buffer has the message buffer number *r* and has following configuration

Table 495. Receive Buffer Configuration

Register	Field	Value	Description
FR_MBCCSR*r*	MTD	0	receive buffer
FR_MBCCFR*r*	MTM	-	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6,...}
	CCFVAL	000000	
FR_MBFIDR*r*	FID	S	subscribed slot

Furthermore the assumption is that both message buffers are enabled (FR_MBCCSR*t*[EDS] = 1 and FR_MBCCSR*r*[EDS] = 1)

Note: The cycle set {4n+2} = {2,6,10,...} is assigned to the receive buffer only.

Note: The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

29.7.7.2 Behavior in static segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S.* In any case, the result of the message buffer search will be the transmit message buffer *t.* The receive message buffer *r* will not be found, no reception is possible.

29.7.7.3 Behavior in dynamic segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

29.7.7.3.1 Transmit Data Not Available

If transmit data are *not available*, i.e. the transmit buffer is not committed FR_MBCCSR*t*[CMT]=0 and/or locked FR_MBCCSR*t*[LCKS]=1,

a) for the cycles in the set {4n}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) for the cycles in the set {4n+2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in *Figure 569*.

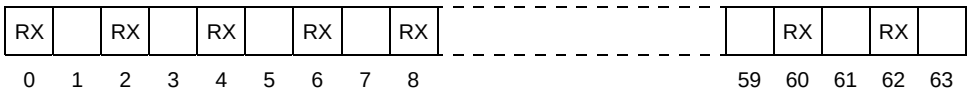


Figure 569. Transmit Data Not Available

29.7.7.3.2 Transmit Data Available

If transmit data are *available,* i.e. the transmit buffer is committed FR_MBCCSR*t*[CMT]=1 and not locked FR_MBCCSR*t*[LCKS]=0,

a) for the cycles in the set {4n}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) for the cycles in the set {4n+2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in *Figure 570*.

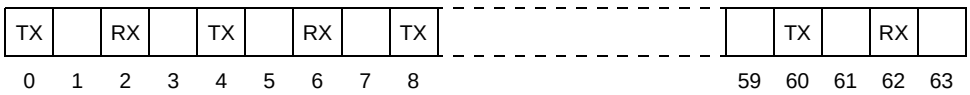


Figure 570. Transmit Data Available

30 Fast Ethernet Controller (FEC)

This chapter provides a feature-set overview, a functional block diagram, and transceiver connection information for both the 10 and 100 Mbit/s MII (Media Independent Interface), as well as the 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

30.1 Overview

The Ethernet Media Access Controller (MAC) is designed to support both 10 and 100 Mbit/s Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FEC supports three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbit/s MII and the 10 Mbit/s-only 7-wire interface, which uses a subset of the MII pins.

30.1.1 Features

The FEC incorporates the following features:

- Support for three different Ethernet physical interfaces:
  - 100-Mbit/s IEEE 802.3 MII
  - 10-Mbit/s IEEE 802.3 MII
  - 10-Mbit/s 7-wire interface (industry standard)
- IEEE 802.3 full duplex flow control
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- Support for full-duplex operation (200 Mbit/s throughput) with a minimum system clock rate of 50MHz (see also *Section 7.4.1.2: FEC Clock Divider Configuration Register (CGM_FEC_DCR)*)
- Support for half-duplex operation (100 Mbit/s throughput) with a minimum system clock rate of 25 MHz (see also *Section 7.4.1.2: FEC Clock Divider Configuration Register (CGM_FEC_DCR)*)
- Retransmission from transmit FIFO following a collision (no processor bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor bus utilization)
- Address recognition
  - Frames with broadcast address may be always accepted or always rejected
  - Exact match for single 48-bit individual (unicast) address
  - Hash (64-bit hash) check of individual (unicast) addresses
  - Hash (64-bit hash) check of group (multicast) addresses
  - Promiscuous mode

30.2 Modes of Operation

The primary operational modes are described in this section.

30.2.1 Full and Half Duplex Operation

Full duplex mode is intended for use on point to point links between switches or end node to switch. Half duplex mode is used in connections between an end node and a repeater or between repeaters. Selection of the duplex mode is controlled by TCR[FDEN].

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC_PAUSE] and TCR[TFC_PAUSE] bits, the RCR[FCE] bit, and *Section 30.4.10: Full Duplex Flow Control*, for more details.

30.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in *Section 30.4.5: Network Interface Options*.

30.2.2.1 10 Mbit/s and 100 Mbit/s MII Interface

MII is the Media Independent Interface defined by the IEEE 802.3 standard for 10/100 Mbit/s operation. The MAC-PHY interface may be configured to operate in MII mode by asserting RCR[MII_MODE].

The speed of operation is determined by the ETXCLK and ERXCLK pins which are driven by the external transceiver. The transceiver will either autonegotiate the speed or it may be controlled by software via the serial management interface (EMDC/EMDIO pins) to the transceiver. Refer to the MMFR and MSCR register descriptions as well as the section on the MII for a description of how to read and write registers in the transceiver via this interface.

30.2.2.2 10 Mbit/s 7-Wire Interface Operation

The FEC supports a 7-wire interface as used by many 10 Mbit/s ethernet transceivers. The RCR[MII_MODE] bit controls this functionality. If this bit is deasserted, the MII mode is disabled and the 10 Mbit/s, 7-wire mode is enabled.

30.2.3 Address Recognition Options

The address options supported are promiscuous, broadcast reject, individual address (hash or exact match), and multicast hash match. Address recognition options are discussed in detail in *Section 30.4.8: Ethernet Address Recognition*.

30.2.4 Internal Loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in *Section 30.4.13: Internal and External Loopback*.

30.3 FEC Top-Level Functional Diagram

The block diagram of the FEC is shown below. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.

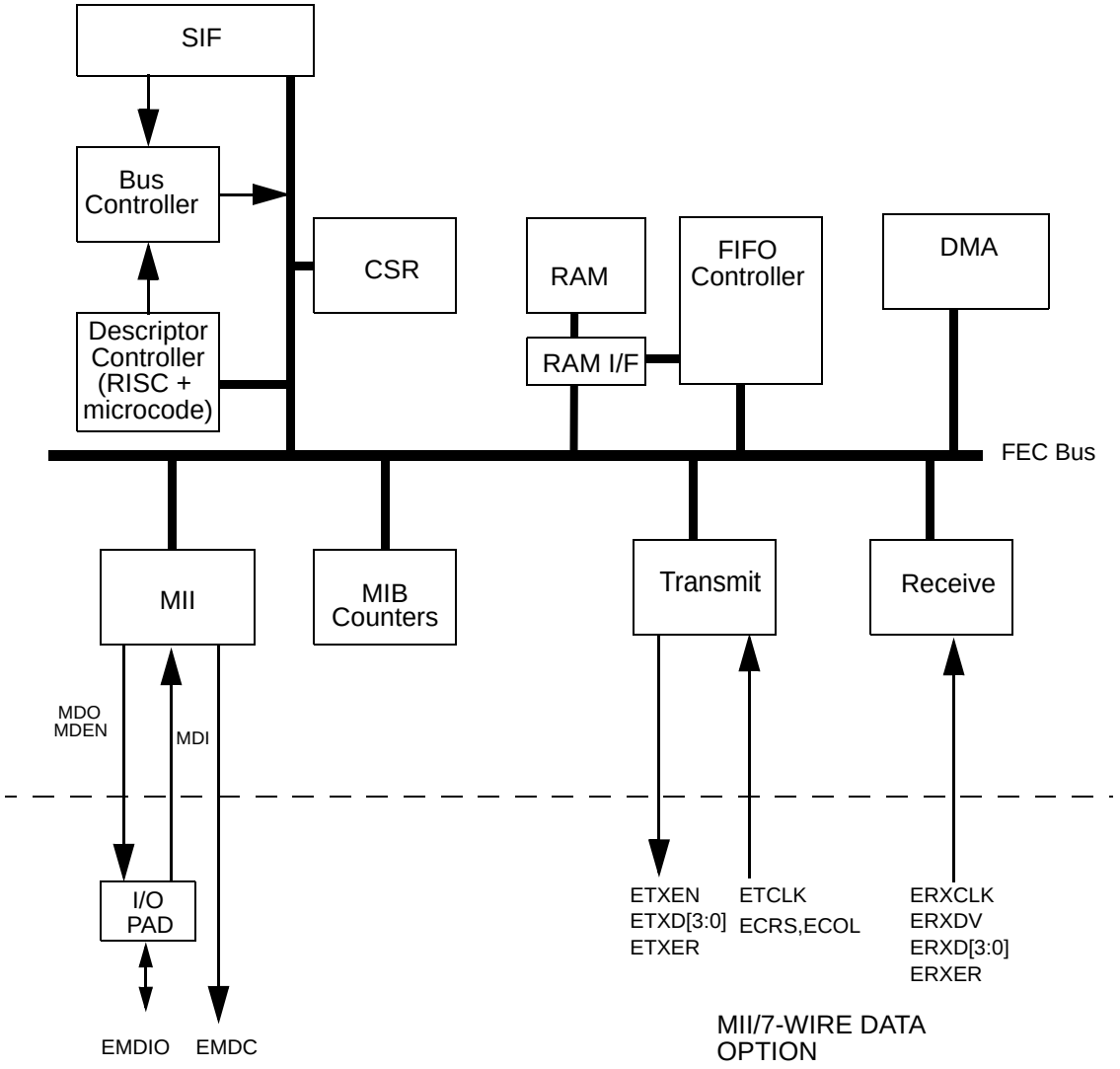


Figure 571. FEC Block Diagram

The descriptor controller is a RISC-based controller that provides the following functions in the FEC:

- Initialization (those internal registers not initialized by the user or hardware)
- High level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

Note: DMA references in this section refer to the FEC's DMA engine. This DMA engine is for the transfer of FEC data only, and is not related to the DMA controller nor to the DMA timers.

The RAM is the focal point of all data flow in the Fast Ethernet Controller and is divided into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR

register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block and receive data flows from the receive block into the receive FIFO.

The user controls the FEC by writing, through the SIF (Slave Interface) module, into control registers located in each block. The CSR (Control and Status Register) block provides global control (e.g. Ethernet reset and enable) and interrupt handling registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the EMDC (Management Data Clock) and EMDIO (Management Data Input/Output) lines of the MII interface.

The DMA block provides multiple channels allowing transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

The Transmit and Receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The Message Information Block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters. See *Section 30.5.3: MIB Block Counters Memory Map*, for more information.

30.4 Functional Description

This section describes the operation of the FEC, beginning with the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FEC.

30.4.1 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations the user must initialize prior to enabling the FEC.

30.4.1.1 Hardware Controlled Initialization

In the FEC, registers and control logic that generate interrupts are reset by hardware. A hardware reset deasserts output signals and resets general configuration bits.

Other registers reset when the ECR[ETHER_EN] bit is cleared. ECR[ETHER_EN] is deasserted by a hard reset or may be deasserted by software to halt operation. By deasserting ECR[ETHER_EN], the configuration control registers such as the TCR and RCR will not be reset, but the entire data path will be reset.

Table 496. ECR[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated

Table 496. ECR[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
RDAR	Cleared
TDAR	Cleared
Descriptor Controller block	Halt operation

30.4.2 User Initialization (Prior to Asserting ECR[ETHER_EN])

The user needs to initialize portions the FEC prior to setting the ECR[ETHER_EN] bit. The exact values will depend on the particular application. The sequence is not important.

Ethernet MAC registers requiring initialization are defined in *Table 497*.

Table 497. User Initialization (Before ENCTRL[ETHER_EN])

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR
PALR / PAUR (only needed for full duplex flow control)
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)
Clear MIB_RAM (locations IPSBAR + 0x1200-0x12FC)

FEC FIFO/DMA registers that require initialization are defined in *Table 498*.

Table 498. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

30.4.3 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, the hardware is ready for operation.

Table 499 shows microcontroller initialization operations.

Table 499. Microcontroller Initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

30.4.4 User Initialization (After Asserting ECR[ETHER_EN])

After asserting ECR[ETHER_EN], the user can set up the buffer/frame descriptors and write to the TDAR and RDAR. Refer to *Section 30.6: Buffer Descriptors*, for more details.

30.4.5 Network Interface Options

The FEC supports both an MII interface for 10/100 Mbit/s Ethernet and a 7-wire serial interface for 10 Mbit/s Ethernet. The interface mode is selected by the RCR[MII_MODE] bit. In MII mode (RCR[MII_MODE] = 1), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. These signals are shown in *Table 500* below.

Table 500. MII Mode

Signal Description	EMAC pin
Transmit Clock	ETXCLK
Transmit Enable	ETXEN
Transmit Data	ETXD[3:0]
Transmit Error	ETXER
Collision	ECOL
Carrier Sense	ECRS
Receive Clock	ERXCLK
Receive Data Valid	ERXDV
Receive Data	ERXD[3:0]
Receive Error	ERXER
Management Data Clock	EMDC

Table 500. MII Mode (continued)

Signal Description	EMAC pin
Management Data Input/Output	EMDIO

The 7-wire serial mode interface (RCR[MII_MODE] = 0) operates in what is generally referred to as the “AMD” mode. 7-wire mode connections to the external transceiver are shown in *Table 501*.

Table 501. 7-Wire Mode Configuration

SIGNAL DESCRIPTION	EMAC PIN
Transmit Clock	TX_CLK
Transmit Enable	ETXEN
Transmit Data	ETXD[0]
Collision	ECOL
Receive Clock	ERXCLK
Receive Data Valid	ERXDV
Receive Data	ERXD[0]

30.4.6 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. Once ECR[ETHER_EN] is asserted and data appears in the transmit FIFO, the Ethernet MAC is able to transmit onto the network.

When the transmit FIFO fills to the watermark (defined by the TFWR), the MAC transmit logic will assert ETXEN and start transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (ECRS asserts). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, the transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See *Section 30.4.14.1: Transmission Errors*, for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data has been transmitted, the FCS (Frame Check Sequence or 32-bit Cyclic Redundancy Check, CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC will be appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Short frames are automatically padded by the transmit logic (if the TC bit in the transmit buffer descriptor for the end of frame buffer = 1).

Both buffer (TXB) and frame (TFINT) interrupts may be generated as determined by the settings in the EIMR.

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL bytes the BABT interrupt will be asserted, however the entire frame will be transmitted (no truncation).

To pause transmission, set the GTS (graceful transmit stop) bit in the TCR register. When the TCR[GTS] is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame either finishes or terminates with a collision. After the transmitter has stopped the GRA (graceful stop complete) interrupt is asserted. If TCR[GTS] is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit first.

30.4.7 FEC Frame Reception

The FEC receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking.

When the driver enables the FEC receiver by asserting ECR[ETHER_EN], it will immediately start processing receive frames. When ERXDV asserts, the receiver will first check for a valid PA/SFD header. If the PA/SFD is valid, it will be stripped and the frame will be processed by the receiver. If a valid PA/SFD is not found, the frame will be ignored.

In serial mode, the first 16 bit times of RX_D0 following assertion of ERXDV are ignored. Following the first 16 bit times the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame have been received, the FEC performs address recognition on the frame.

Once a collision window (64 bytes) of data has been received and if address recognition has not rejected the frame, the receive FIFO is signalled that the frame is “accepted” and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to “reject” the frame. Thus, no collision fragments are presented to the user except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and once the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV and TR status bits, and the frame length. See *Section 30.4.14.2: Reception Errors*, for more details.

Receive Buffer (RXB) and Frame Interrupts (RFINT) may be generated if enabled by the EIMR register. A receive error interrupt is babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt will occur and the LG bit in the Receive Buffer Descriptor (RxBD) will be set. See *Section 30.6.2: Ethernet Receive Buffer Descriptor (RxBD)*, for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR, maskable by RFIEN bit in EIMR), indicating that a

frame has been received and is in memory. The Ethernet controller then waits for a new frame.

The Ethernet controller receives serial data LSB first.

30.4.8 Ethernet Address Recognition

The FEC filters the received frames based on destination address (DA) type — individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames is illustrated in the figures below.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in *Figure 572* illustrates the address recognition decisions made by the receive block, while *Figure 573* illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC_REJ]) is deasserted, then the frame will be accepted unconditionally, as shown in *Figure 572*. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in *Figure 573*.

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller will perform a group hash table lookup using the 64-entry hash table programmed in GAUR and GALR. If a hash match occurs, the receiver accepts the frame.

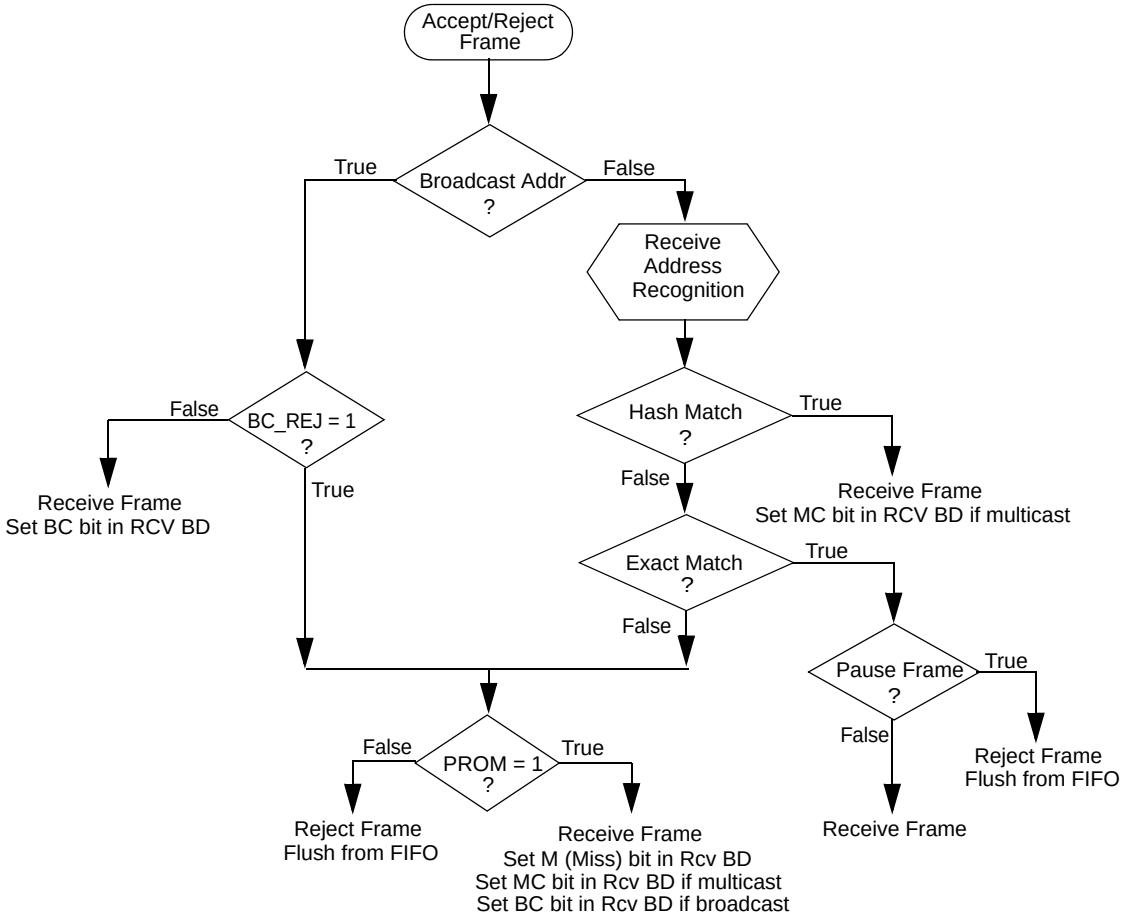
If flow control is enabled, the microcontroller will do an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines that the received frame is a valid PAUSE frame, then the frame will be rejected. Note the receiver will detect a PAUSE frame with the DA field set to either the designated PAUSE DA or the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that the user programs in the PALR and PAUR registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR and IALR. In the case of an individual hash match, the frame is accepted. Again, the receiver will accept or reject the frame based on PAUSE frame detection, shown in *Figure 572*.

If neither a hash match (group or individual), nor an exact match (group or individual) occur, then if promiscuous mode is enabled (RCR[PROM] = 1), then the frame will be accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame will be rejected.

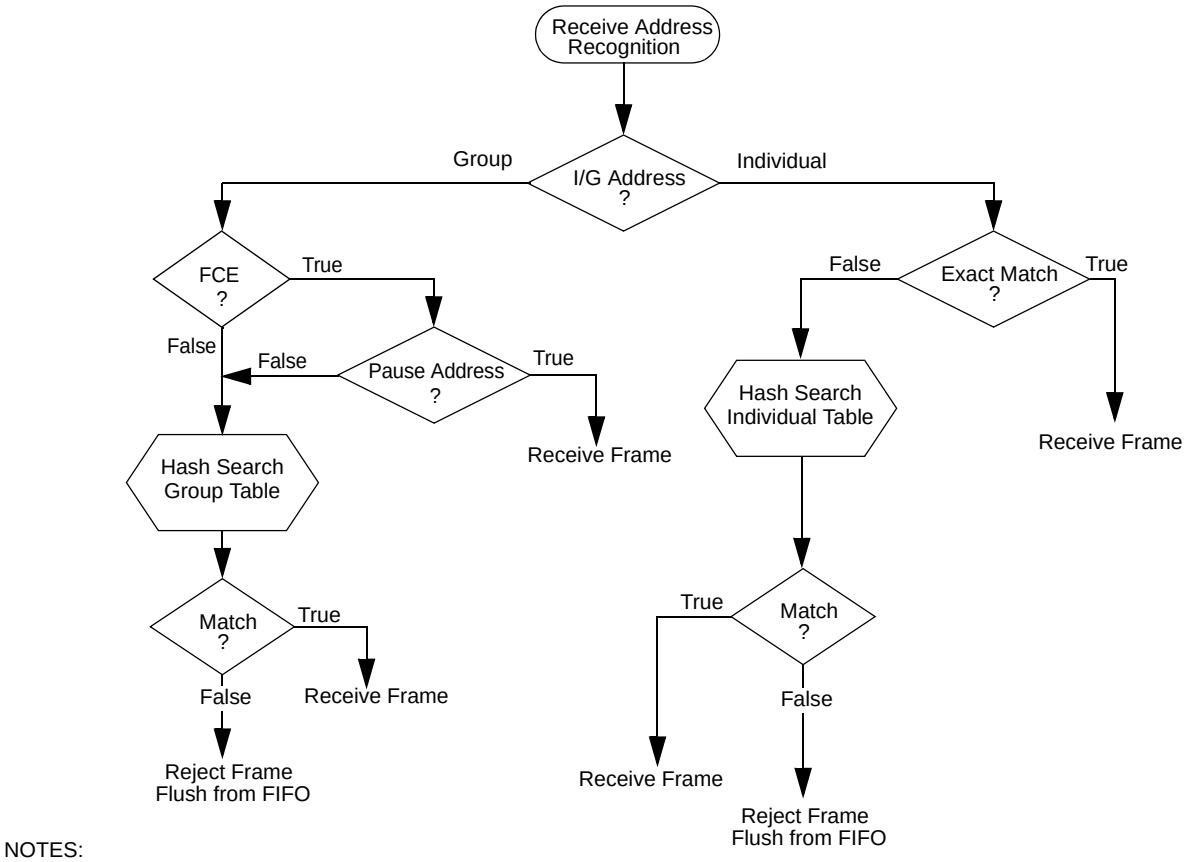
Similarly, if the DA is a broadcast address, broadcast reject (RCR[BC_REJ]) is asserted, and promiscuous mode is enabled, then the frame will be accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame will be rejected.

In general, when a frame is rejected, it is flushed from the FIFO.



NOTES:
BC_REJ - field in RCR register (BroadCast REJect)
PROM - field in RCR register (PROMiscous mode)
Pause Frame - valid PAUSE frame received

Figure 572. Ethernet Address Recognition—Receive Block Decisions



NOTES:
FCE - field in RCR register (Flow Control Enable)
I/G - Individual/Group bit in Destination Address (least significant bit in first byte received in MAC frame)

Figure 573. Ethernet Address Recognition—Microcode Decisions

30.4.9 Hash Algorithm

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, which are represented by 64 bits stored in GAUR, GALR (group address hash match) or IAUR, IALR (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the 6 most significant bits of the CRC-encoded result to generate a number between 0 and 63. The MSB of the CRC result selects GAUR (MSB = 1) or GALR (MSB = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit that is set in the hash table, the frame is accepted; otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The hash table registers must be initialized by the user. The CRC32 polynomial to use in computing the hash is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

A table of example Destination Addresses and corresponding hash values is included below for reference.

Table 502. Destination Address to 6-Bit Hash

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65:ff:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff:ff	0x3	3
b5:ff:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff:ff	0x5	5
d5:ff:ff:ff:ff:ff	0x6	6
f5:ff:ff:ff:ff:ff	0x7	7
db:ff:ff:ff:ff:ff	0x8	8
fb:ff:ff:ff:ff:ff	0x9	9
bb:ff:ff:ff:ff:ff	0xa	10
8b:ff:ff:ff:ff:ff	0xb	11
0b:ff:ff:ff:ff:ff	0xc	12
3b:ff:ff:ff:ff:ff	0xd	13
7b:ff:ff:ff:ff:ff	0xe	14
5b:ff:ff:ff:ff:ff	0xf	15
27:ff:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff:ff	0x13	19
f7:ff:ff:ff:ff:ff	0x14	20
c7:ff:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff:ff	0x16	22
a7:ff:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff:ff	0x18	24
b9:ff:ff:ff:ff:ff	0x19	25
f9:ff:ff:ff:ff:ff	0x1a	26
c9:ff:ff:ff:ff:ff	0x1b	27
59:ff:ff:ff:ff:ff	0x1c	28

Table 502. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
79:ff:ff:ff:ff:ff	0x1d	29
29:ff:ff:ff:ff:ff	0x1e	30
19:ff:ff:ff:ff:ff	0x1f	31
d1:ff:ff:ff:ff:ff	0x20	32
f1:ff:ff:ff:ff:ff	0x21	33
b1:ff:ff:ff:ff:ff	0x22	34
91:ff:ff:ff:ff:ff	0x23	35
11:ff:ff:ff:ff:ff	0x24	36
31:ff:ff:ff:ff:ff	0x25	37
71:ff:ff:ff:ff:ff	0x26	38
51:ff:ff:ff:ff:ff	0x27	39
7f:ff:ff:ff:ff:ff	0x28	40
4f:ff:ff:ff:ff:ff	0x29	41
1f:ff:ff:ff:ff:ff	0x2a	42
3f:ff:ff:ff:ff:ff	0x2b	43
bf:ff:ff:ff:ff:ff	0x2c	44
9f:ff:ff:ff:ff:ff	0x2d	45
df:ff:ff:ff:ff:ff	0x2e	46
ef:ff:ff:ff:ff:ff	0x2f	47
93:ff:ff:ff:ff:ff	0x30	48
b3:ff:ff:ff:ff:ff	0x31	49
f3:ff:ff:ff:ff:ff	0x32	50
d3:ff:ff:ff:ff:ff	0x33	51
53:ff:ff:ff:ff:ff	0x34	52
73:ff:ff:ff:ff:ff	0x35	53
23:ff:ff:ff:ff:ff	0x36	54
13:ff:ff:ff:ff:ff	0x37	55
3d:ff:ff:ff:ff:ff	0x38	56
0d:ff:ff:ff:ff:ff	0x39	57
5d:ff:ff:ff:ff:ff	0x3a	58
7d:ff:ff:ff:ff:ff	0x3b	59
fd:ff:ff:ff:ff:ff	0x3c	60
dd:ff:ff:ff:ff:ff	0x3d	61
9d:ff:ff:ff:ff:ff	0x3e	62
bd:ff:ff:ff:ff:ff	0x3f	63

30.4.10 Full Duplex Flow Control

Full-duplex flow control allows the user to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable pause frame detection, the FEC must operate in full-duplex mode (TCR[FDEN] asserted) and flow control enable (RCR[FCE]) must be asserted. The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in the table below. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 503. PAUSE Frame Field Specification

48-bit Destination Address	0x0180_c200_0001 or Physical Address
48-bit Source Address	Any
16-bit Type	0x8808
16-bit Opcode	0x0001
16-bit PAUSE Duration	0x0000 to 0xFFFF

Pause frame detection is performed by the receiver and microcontroller modules. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, TCR[GTS] is asserted by the FEC internally. When transmission has paused, the EIR[GRA] interrupt is asserted and the pause timer begins to increment. Note that the pause timer makes use of the transmit backoff timer hardware, which is used for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD[PAUSE_DUR] slot times have expired. On OPD[PAUSE_DUR] expiration, TCR[GTS] is deasserted allowing MAC data frame transmission to resume. Note that the receive flow control pause (TCR[RFC_PAUSE]) status bit is asserted while the transmitter is paused due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and the user must assert flow control pause (TCR[TFC_PAUSE]). On assertion of transmit flow control pause (TCR[TFC_PAUSE]), the transmitter asserts TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts. Following EIR[GRA] assertion, the pause frame is transmitted. On completion of pause frame transmission, flow control pause (TCR[TFC_PAUSE]) and TCR[GTS] are deasserted internally.

The user must specify the desired pause duration in the OPD register.

Note that when the transmitter is paused due to receiver/microcontroller pause frame detection, transmit flow control pause (TCR[TFC_PAUSE]) still may be asserted and will cause the transmission of a single pause frame. In this case, the EIR[GRA] interrupt will not be asserted.

30.4.11 Inter-Packet Gap (IPG) Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit

times. If carrier sense asserts during the last 36 bit times, it will be ignored and a collision will occur.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the following frame may be discarded by the receiver.

30.4.12 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller will continue the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern will be sent after the end of the preamble sequence.

If a collision occurs within 512 bit times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 512 bit times, then no retransmission is performed and the end of frame buffer is closed with a Late Collision (LC) error indication.

30.4.13 Internal and External Loopback

Both internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Both internal and external loopback are configured using combinations of the LOOP and DRT bits in the RCR register and the FDEN bit in the TCR register.

For both internal and external loopback set FDEN = 1.

For internal loopback set RCR[LOOP] = 1 and RCR[DRT] = 0. ETXEN and ETXER will not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the internal system clock is used by the transmit and receive blocks instead of the clocks from the external transceiver. This will cause an increase in the required system bus bandwidth for transmit and receive data being DMA'd to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback set RCR[LOOP] = 0, RCR[DRT] = 0 and configure the external transceiver for loopback.

30.4.14 Ethernet Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs, the EIR register, and the MIB block counters.

30.4.14.1 Transmission Errors

30.4.14.1.1 Transmitter Underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed. The UN bit is set in the EIR. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The “UN” interrupt will be asserted if enabled in the EIMR register.

30.4.14.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the RL bit is set in the EIR. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The “RL” interrupt will be asserted if enabled in the EIMR register.

30.4.14.1.3 Late Collision

When a collision occurs after the slot time (512 bits starting at the Preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the LC bit is set in the EIR register. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The “LC” interrupt will be asserted if enabled in the EIMR register.

30.4.14.1.4 Heartbeat

Some transceivers have a self-test feature called “heartbeat” or “signal quality error.” To signify a good self-test, the transceiver indicates a collision to the FEC within 4 microseconds after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition.

If the HBC bit is set in the TCR register and the heartbeat condition is not detected by the **FEC** after a frame transmission, then a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets the HB bit in the EIR register, and generates the HBERR interrupt if it is enabled.

30.4.14.2 Reception Errors

30.4.14.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, the FEC sets the OV bit in the RxBD. All subsequent data in the frame will be discarded and subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. This frame must be discarded by the driver.

30.4.14.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller handles up to seven dribbling bits when the receive frame terminates past an non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If there is a CRC error, then the frame non-octet aligned (NO) error is reported in the RxBD. If there is no CRC error, then no error is reported.

30.4.14.2.3 CRC Error

When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

30.4.14.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes the BABR interrupt will be generated, and the LG bit in the end of frame RxBD will be set. The frame is not truncated unless the frame length exceeds 2047 bytes).

30.4.14.2.5 Truncation

When the receive frame length exceeds 2047 bytes the frame is truncated and the TR bit is set in the receive BD.

30.5 Programming Model

This section gives an overview of the registers, followed by a description of the buffers.

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control and to extract global status information. The descriptors are used to pass data buffers and related buffer information between the hardware and software.

30.5.1 Top Level Module Memory Map

The FEC implementation requires a 1-Kbyte memory map space. This is divided into 2 sections of 512 bytes each. The first is used for control/status registers. The second contains event/statistic counters held in the MIB block. *Table 504* defines the top level memory map.

Table 504. Module Memory Map

Address	Function
IPSBAR + 0x1000-11FF	Control/Status Registers
IPSBAR + 0x1200-13FF	MIB Block Counters

30.5.2 Register map

Table 505 shows the FEC register map. All offsets not explicitly mentioned are reserved.

Table 505. FEC register map

Base address: 0xFFF4_C000		
Address offset	**Register**	**Location**
0x1004	Ethernet Interrupt Event Register (EIR)	*on page 984*
0x1008	Interrupt Mask Register (EIMR)	*on page 986*
0x1010	Receive Descriptor Active Register (RDAR)	*on page 987*
0x1014	Transmit Descriptor Active Register (TDAR)	*on page 987*
0x1024	Ethernet Control Register (ECR)	*on page 988*
0x1040	MII Management Frame Register (MMFR)	*on page 989*
0x1044	MII Speed Control Register (MSCR)	*on page 990*

Table 505. FEC register map (continued)

Base address: 0xFFF4_C000		
Address offset	**Register**	**Location**
0x1064	MIB Control Register (MIBC)	*on page 991*
0x1084	Receive Control Register (RCR)	*on page 992*
0x10C4	Transmit Control Register (TCR)	*on page 993*
0x10E4	Physical Address Low Register (PALR)	*on page 993*
0x10E8	Physical Address High Register (PAUR)	*on page 994*
0x10EC	Opcode + Pause Duration Register (OPD)	*on page 994*
0x1118	Descriptor Individual Upper Address Register (IAUR)	*on page 995*
0x111C	Descriptor Individual Lower Address Register (IALR)	*on page 995*
0x1120	Descriptor Group Upper Address Register (GAUR)	*on page 996*
0x1124	Descriptor Group Lower Address Register (GALR)	*on page 996*
0x1144	FIFO Transmit FIFO Watermark Register (TFWR)	*on page 997*
0x114C	FIFO Receive Bound Register (FRBR)	*on page 997*
0x1150	FIFO Receive Start Register (FRSR)	*on page 998*
0x1180	Receive Descriptor Ring Start Register (ERDSR)	*on page 998*
0x1184	Transmit Buffer Descriptor Ring Start (ETDSR)	*on page 999*
0x1188	Receive Buffer Size Register (EMRBR)	*on page 999*

30.5.3 MIB Block Counters Memory Map

Table 506 defines the MIB Counters memory map which defines the locations in the MIB RAM space where hardware maintained counters reside. These fall in the 0x1200-0x13FF address offset range. The counters are divided into two groups.

RMON counters are included which cover the Ethernet Statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet Statistics group, a counter is included to count truncated frames as the FEC only supports frame lengths up to 2047 bytes. The RMON counters are implemented independently for transmit and receive to insure accurate network statistics when operating in full duplex mode.

IEEE counters are included which support the Mandatory and Recommended counter packages defined in section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The IEEE Basic Package objects are supported by the FEC but do not require counters in the MIB block. In addition, some of the recommended package objects which are supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are included as well.

Table 506. MIB Counters Memory Map

Address Offset	Mnemonic	Description
0x1200	RMON_T_DROP	Count of frames not counted correctly

Table 506. MIB Counters Memory Map (continued)

Address Offset	Mnemonic	Description
0x1204	RMON_T_PACKETS	RMON Tx packet count
0x1208	RMON_T_BC_PKT	RMON Tx Broadcast Packets
0x120C	RMON_T_MC_PKT	RMON Tx Multicast Packets
0x1210	RMON_T_CRC_ALIGN	RMON Tx Packets w CRC/Align error
0x1214	RMON_T_UNDERSIZE	RMON Tx Packets < 64 bytes, good crc
0x1218	RMON_T_OVERSIZE	RMON Tx Packets > MAX_FL bytes, good crc
0x121C	RMON_T_FRAG	RMON Tx Packets < 64 bytes, bad crc
0x1220	RMON_T_JAB	RMON Tx Packets > MAX_FL bytes, bad crc
0x1224	RMON_T_COL	RMON Tx collision count
0x1228	RMON_T_P64	RMON Tx 64 byte packets
0x122C	RMON_T_P65TO127	RMON Tx 65 to 127 byte packets
0x1230	RMON_T_P128TO255	RMON Tx 128 to 255 byte packets
0x1234	RMON_T_P256TO511	RMON Tx 256 to 511 byte packets
0x1238	RMON_T_P512TO1023	RMON Tx 512 to 1023 byte packets
0x123C	RMON_T_P1024TO2047	RMON Tx 1024 to 2047 byte packets
0x1240	RMON_T_P_GTE2048	RMON Tx packets w > 2048 bytes
0x1244	RMON_T_OCTETS	RMON Tx Octets
0x1248	IEEE_T_DROP	Count of frames not counted correctly
0x124C	IEEE_T_FRAME_OK	Frames Transmitted OK
0x1250	IEEE_T_1COL	Frames Transmitted with Single Collision
0x1254	IEEE_T_MCOL	Frames Transmitted with Multiple Collisions
0x1258	IEEE_T_DEF	Frames Transmitted after Deferral Delay
0x125c	IEEE_T_LCOL	Frames Transmitted with Late Collision
0x1260	IEEE_T_EXCOL	Frames Transmitted with Excessive Collisions
0x1264	IEEE_T_MACERR	Frames Transmitted with Tx FIFO Underrun
0x1268	IEEE_T_CSERR	Frames Transmitted with Carrier Sense Error
0x126C	IEEE_T_SQE	Frames Transmitted with SQE Error
0x1270	IEEE_T_FDXFC	Flow Control Pause frames transmitted
0x1274	IEEE_T_OCTETS_OK	Octet count for Frames Transmitted w/o Error
0x1284	RMON_R_PACKETS	RMON Rx packet count
0x1288	RMON_R_BC_PKT	RMON Rx Broadcast Packets
0x128C	RMON_R_MC_PKT	RMON Rx Multicast Packets
0x1290	RMON_R_CRC_ALIGN	RMON Rx Packets w CRC/Align error
0x1294	RMON_R_UNDERSIZE	RMON Rx Packets < 64 bytes, good crc

Table 506. MIB Counters Memory Map (continued)

Address Offset	Mnemonic	Description
0x1298	RMON_R_OVERSIZE	RMON Rx Packets > MAX_FL bytes, good crc
0x129C	RMON_R_FRAG	RMON Rx Packets < 64 bytes, bad crc
0x12A0	RMON_R_JAB	RMON Rx Packets > MAX_FL bytes, bad crc
0x12A4	RMON_R_RESVD_0	
0x12A8	RMON_R_P64	RMON Rx 64 byte packets
0x12AC	RMON_R_P65TO127	RMON Rx 65 to 127 byte packets
0x12B0	RMON_R_P128TO255	RMON Rx 128 to 255 byte packets
0x12B4	RMON_R_P256TO511	RMON Rx 256 to 511 byte packets
0x12B8	RMON_R_P512TO1023	RMON Rx 512 to 1023 byte packets
0x12BC	RMON_R_P1024TO2047	RMON Rx 1024 to 2047 byte packets
0x12C0	RMON_R_P_GTE2048	RMON Rx packets w > 2048 bytes
0x12C4	RMON_R_OCTETS	RMON Rx Octets
0x12C8	IEEE_R_DROP	Count of frames not counted correctly
0x12CC	IEEE_R_FRAME_OK	Frames Received OK
0x12D0	IEEE_R_CRC	Frames Received with CRC Error
0x12D4	IEEE_R_ALIGN	Frames Received with Alignment Error
0x12D8	IEEE_R_MACERR	Receive Fifo Overflow count
0x12DC	IEEE_R_FDXFC	Flow Control Pause frames received
0x12E0	IEEE_R_OCTETS_OK	Octet count for Frames Rcvd w/o Error

30.5.4 Registers

The following sections describe each register in detail.

30.5.4.1 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in the EIR register, an interrupt will be generated if the corresponding bit in the interrupt mask register (EIMR) is also set. The bit in the EIR register is cleared if a one is written to that bit position; writing zero has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts which may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC and RL. Interrupts resulting from internal errors are HBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts since these errors will be visible to network management via the MIB counters.

- HBERR - IEEE_T_SQE
- BABR - RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT - RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)
- LATE_COL - IEEE_T_LCOL
- COL_RETRY_LIM - IEEE_T_EXCOL
- XFIFO_UN - IEEE_T_MACERR

Offset: 0x004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EBERR	LC	RL	UN	0	0	0
W	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	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

Figure 574. Ethernet Interrupt Event Register (EIR)

Table 507. EIR field descriptions

Field	Description
HBERR	Heartbeat error. This interrupt indicates that HBC is set in the TCR register and that the COL input was not asserted within the Heartbeat window following a transmission.
BABR	Babbling receive error. This bit indicates a frame was received with length in excess of RCR[MAX_FL] bytes.
BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded RCR[MAX_FL] bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffer(s). Truncation does not occur.
GRA	Graceful stop complete. This interrupt will be asserted for one of three reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. 1) A graceful stop, which was initiated by the setting of the TCR[GTS] bit is now complete. 2) A graceful stop, which was initiated by the setting of the TCR[TFC_PAUSE] bit is now complete. 3) A graceful stop, which was initiated by the reception of a valid full duplex flow control “pause” frame is now complete. Refer to the “Full Duplex Flow Control” section of the Functional Description chapter.
TXF	Transmit frame interrupt. This bit indicates that a frame has been transmitted and that the last corresponding buffer descriptor has been updated.
TXB	Transmit buffer interrupt. This bit indicates that a transmit buffer descriptor has been updated.

Table 507. EIR field descriptions (continued)

Field	Description
RXF	Receive frame interrupt. This bit indicates that a frame has been received and that the last corresponding buffer descriptor has been updated.
RXB	Receive buffer interrupt. This bit indicates that a receive buffer descriptor has been updated that was not the last in the frame.
MII	MII interrupt. This bit indicates that the MII has completed the data transfer requested.
EBERR	Ethernet bus error. This bit indicates that a system bus error occurred when a DMA transaction was underway. When the EBERR bit is set, ECR[ETHER_EN] will be cleared, halting frame processing by the FEC. When this occurs software will need to insure that the FIFO controller and DMA are also soft reset.
LC	Late collison. This bit indicates that a collision occurred beyond the collision window (slot time) in half duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.
RL	Collision retry limit. This bit indicates that a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted and transmission of the next frame will commence. Can only occur in half duplex mode.
UN	Transmit FIFO underrun. This bit indicates that the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.

30.5.4.2 Interrupt Mask Register (EIMR)

The EIMR register controls which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. This register is cleared upon a hardware reset. If the corresponding bits in both the EIR and EIMR registers are set, the interrupt will be signalled to the CPU. The interrupt signal will remain asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

Offset: 0x008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EBERR	LC	RL	UN	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

Figure 575. Ethernet Interrupt Mask Register (EIMR)

Table 508. EIMR field descriptions

Name	Description
See *Figure 574* and *Table 507*.	Interrupt Mask. Each bit corresponds to an interrupt source defined by the EIR register. The corresponding EIMR bit determines whether an interrupt condition can generate an interrupt. At every clock, the EIR samples the signal generated by the interrupting source. The corresponding EIR bit reflects the state of the interrupt signal even if the corresponding EIMR bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.

30.5.4.3 Receive Descriptor Active Register (RDAR)

RDAR is a command register, written by the user, that indicates that the receive descriptor ring has been updated (empty receive buffers have been produced by the driver with the empty bit set).

Whenever the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC will poll the receive descriptor ring and process receive frames (provided ECR[ETHER_EN] is also set). Once the FEC polls a receive descriptor whose empty bit is not set, then the FEC will clear the RDAR bit and cease receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors have been placed into the receive descriptor ring.

The RDAR register is cleared at reset and when ECR[ETHER_EN] is cleared.

Offset: 0x010 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	R_DES_ACTIVE	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																																
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

Figure 576. Receive Descriptor Active Register (RDAR)

Table 509. RDAR field descriptions

Field	Description
R_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “empty” descriptors remain in the receive ring. Also cleared when ECR[ETHER_EN] is cleared.

30.5.4.4 Transmit Descriptor Active Register (TDAR)

The TDAR is a command register which should be written by the user to indicate that the transmit descriptor ring has been updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

Whenever the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the FEC will poll the transmit descriptor ring and process transmit frames (provided ECR[ETHER_EN] is also set). Once the FEC polls a

transmit descriptor whose ready bit is not set, then the FEC will clear the TDAR bit and cease transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The TDAR register is cleared at reset, when ECR[ETHER_EN] is cleared, or when ECR[RESET] is set.

Offset: 0x014 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	X_DES_ACTIVE	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																																
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

Figure 577. Transmit Descriptor Active Register (TDAR)

Table 510. TDAR Field Descriptions

Name	Description
X_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “ready” descriptors remain in the transmit ring. Also cleared when ECR[ETHER_EN] is cleared.

30.5.4.5 Ethernet Control Register (ECR)

ECR is a read/write user register, though both fields in this register may be altered by hardware as well. The ECR is used to enable/disable the FEC.

Offset: 0x024 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	ETHER_EN	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 578. Ethernet Control Register (ECR)

Table 511. ECR field descriptions

Field	Description
ETHER_EN	When this bit is set, the FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception is immediately stopped and transmission is stopped after a bad CRC is appended to any currently transmitted frame. The buffer descriptor(s) for an aborted transmit frame are not updated after clearing this bit. When ETHER_EN is deasserted, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. The ETHER_EN bit is altered by hardware under the following conditions: – ECR[RESET] is set by software, in which case ETHER_EN will be cleared – an error condition causes the EIR[EBERR] bit to set, in which case ETHER_EN will be cleared
RESET	When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. ETHER_EN is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately 8 clock cycles after RESET is written with a 1.

30.5.4.6 MII Management Frame Register (MMFR)

The MMFR register is accessed by the user and does not reset to a defined value. The MMFR register is used to communicate with the attached MII compatible PHY device(s), providing read/write access to their MII registers. Performing a write to the MMFR register will cause a management frame to be sourced unless the MSCR register has been programmed to 0. In the case of writing to MMFR when MSCR = 0, if the MSCR register is then written to a non-zero value, an MII frame will be generated with the data previously written to the MMFR register. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.

To perform a read or write operation on the MII Management Interface, the MMFR register must be written by the user. To generate a valid read or write management frame, the ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame will be generated but will not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII Management Interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR register. Writing this pattern will cause the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time the contents of the MMFR register will be altered as the contents are serially shifted and will be unpredictable if read by the user. Once the write management frame operation has completed, the MII interrupt will be generated. At this time the contents of the MMFR register will match the original value written.

To generate an MII Management Interface read frame (read a PHY register) the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR register (the content of the DATA field is a don't care). Writing this pattern will cause the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time the contents of the MMFR register will be altered as the contents are serially shifted, and will be unpredictable if read by the user. Once the read management frame operation has completed, the MII interrupt will be generated. At this time the contents of the MMFR register will match the original value written except for the DATA field whose contents have been replaced by the value read from the PHY register.

If the MMFR register is written while frame generation is in progress, the frame contents will be altered. Software should use the MII_STATUS register and/or the MII interrupt to avoid writing to the MMFR register while frame generation is in progress.

Offset: 0x040 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 W	ST	OP	PA	RA	TA	DATA		
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 579. MII Management Frame Register (MMFR)

Table 512. MMFR field descriptions

Field	Description
ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
OP	Operation code. This field must be programmed to 10 (read) or 01 (write) to generate a valid MII management frame. A value of 11 will produce “read” frame operation while a value of 00 will produce “write” frame operation, but these frames will not be MII compliant.
PA	PHY address. This field specifies one of up to 32 attached PHY devices.
RA	Register address. This field specifies one of up to 32 registers within the specified PHY device.
TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

30.5.4.7 MII Speed Control Register (MSCR)

The MSCR register provides control of the MII clock (EMDC pin) frequency, allows a preamble drop on the MII management frame, and provides observability (intended for manufacturing test) of an internal counter used in generating the EMDC clock signal.

The MII_SPEED field must be programmed with a value to provide an EMDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII_SPEED must be set to a non-zero value in order to source a read or write management frame. After the management frame is complete the MSCR register may optionally be set to zero to turn off the EMDC. The EMDC generated will have a 50% duty cycle except when MII_SPEED is changed during operation (change will take effect following either a rising or falling edge of EMDC).

If the system clock is 25 MHz, programming this register to 0x0000_0005 will result in an EMDC frequency of 25 MHz * 1/10 = 2.5 MHz. *Table 514* shows the optimum values for MII_SPEED as a function of system clock frequency.

Offset: 0x044 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	DIS_PRE	MII_SPEED	0
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

Figure 580. MII Speed Control Register (MSCR)

Table 513. MSCR field descriptions

Field	Description
DIS_PREAMBLE	Asserting this bit will cause preamble (32 1's) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY device(s) does not require it.
MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (EMDC) relative to system clock. A value of 0 in this field will "turn off" the EMDC and leave it in low voltage state. Any non-zero value will result in the EMDC frequency of 1/(MII_SPEED*2) of the system clock frequency.

Table 514. Programming examples for MSCR

System clock frequency	MII_SPEED (field in reg)	EMDC frequency
25 MHz	0x5	2.5 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.5 MHz
50 MHz	0xA	2.5 MHz
66 MHz	0xD	2.5 MHz

30.5.4.8 MIB Control Register (MIBC)

The MIB control register is a read/write register used to provide control of and to observe the state of the MIB block. This register is accessed by user software if there is a need to disable the MIB block operation. For example, in order to clear all MIB counters in RAM the user should disable the MIB block, then clear all the MIB RAM locations, then enable the MIB block. The MIB_DISABLE bit is reset to 1. See *Table 506* for the locations of the MIB counters.

Offset: 0x064 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	MIB_DIS	MIB_IDLE	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																																
Reset	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 581. MIB Control Register (MIBC)

Table 515. MIBC field descriptions

Field	Description
MIB_DISABLE	A read/write control bit. If set, the MIB logic will halt and not update any MIB counters.
MIB_IDLE	A read-only status bit. If set the MIB block is not currently updating any MIB counters.

30.5.4.9 Receive Control Register (RCR)

The RCR is programmed by the user. The RCR controls the operational mode of the receive block and should be written only when ECR[ETHER_EN] = 0 (initialization time).

Offset: 0x084 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	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 582. Receive Control Register (RCR)

Table 516. RCR field descriptions

Field	Description
MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL will cause the BABT interrupt to occur. Receive Frames longer than MAX_FL will cause the BABR interrupt to occur and will set the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed by the user is 1518 or 1522 (if VLAN Tags are supported).
FCE	Flow control enable. If asserted, the receiver will detect PAUSE frames. Upon PAUSE frame detection, the transmitter will stop transmitting data frames for a given duration.
BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) = FF_FF_FF_FF_FF_FF will be rejected unless the PROM bit is set. If both BC_REJ and PROM = 1, then frames with broadcast DA will be accepted and the M (MISS) bit will be set in the receive buffer descriptor.
PROM	Promiscuous mode. All frames are accepted regardless of address matching.
MII_MODE	Media independent interface mode. Selects external interface mode. Setting this bit to one selects MII mode, setting this bit equal to zero selects 7-wire mode (used only for serial 10 Mbit/s). This bit controls the interface mode for both transmit and receive blocks.
DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting (normally used for half duplex mode).
LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The system clock is substituted for the ETXCLK when LOOP is asserted. DRT must be set to zero when asserting LOOP.

30.5.4.10 Transmit Control Register (TCR)

This register is read/write and is written by the user to configure the transmit block. This register is cleared at system reset. The FDEN and HBC bits should be modified only when ECR[ETHER_EN] = 0.

Offset: 0x0C4 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	RFC_PAUSE	TFC_PAUSE	FDEN	HBC	GTS
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

Figure 583. Transmit Control Register (TCR)

Table 517. TCR field descriptions

Field	Description
RFC_PAUSE	Receive frame control pause. This read-only status bit will be asserted when a full duplex flow control pause frame has been received and the transmitter is paused for the duration defined in this pause frame. This bit will automatically clear when the pause duration is complete.
TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC will stop transmission of data frames after the current transmission is complete. At this time, the GRA interrupt in the EIR register will be asserted. With transmission of data frames stopped, the MAC will transmit a MAC Control PAUSE frame. Next, the MAC will clear the TFC_PAUSE bit and resume transmitting data frames. Note that if the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC may still transmit a MAC Control PAUSE frame.
FDEN	Full duplex enable. If set, frames are transmitted independent of carrier sense and collision inputs. This bit should only be modified when ETHER_EN is deasserted.
HBC	Heartbeat control. If set, the heartbeat check is performed following end of transmission and the HB bit in the status register will be set if the collision input does not assert within the heartbeat window. This bit should only be modified when ETHER_EN is deasserted.
GTS	Graceful transmit stop. When this bit is set, the MAC will stop transmission after any frame that is currently being transmitted is complete and the GRA interrupt in the EIR register will be asserted. If frame transmission is not currently underway, the GRA interrupt will be asserted immediately. Once transmission has completed, a “restart” can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO will then be transmitted. If an early collision occurs during transmission when GTS = 1, transmission will stop after the collision. The frame will be transmitted again once GTS is cleared. Note that there may be old frames in the transmit FIFO that will be transmitted when GTS is reasserted. To avoid this deassert ECR[ETHER_EN] following the GRA interrupt.

30.5.4.11 Physical Address Lower Register (PALR)

The PALR register is written by the user. This register contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit address used in the address recognition process to compare with the DA (Destination Address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the 6-byte Source Address field when transmitting PAUSE frames. This register is not reset and must be initialized by the user.

Offset: 0x0E4 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 W	PADDR1							
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 584. Physical Address Lower Register (PALR)

Table 518. PALR field descriptions

Field	Description
PADDR1	Bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8) and 3 (bits 7:0) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.

30.5.4.12 Physical Address Upper Register (PAUR)

The PAUR register is written by the user. This register contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA (Destination Address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 15:0 of PAUR contain a constant type field (0x8808) used for transmission of PAUSE frames. This register is not reset and bits 31:16 must be initialized by the user.

Offset: 0x0E8 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 W	PADDR2				TYPE			
Reset	— — — —	— — — —	— — — —	— — — —	1 0 0 0	1 0 0 0	0 0 0 0	1 0 0 0

Figure 585. Physical Address Upper Register (PAUR)

Table 519. PAUR field descriptions

Field	Description
PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.
TYPE	Type field in PAUSE frames. These 16-bits are a constant value of 0x8808.

30.5.4.13 Opcode/Pause Duration Register (OPD)

The OPD register is read/write accessible. This register contains the 16-bit Opcode, and 16-bit pause duration fields used in transmission of a PAUSE frame. The Opcode field is a constant value, 0x0001. When another node detects a PAUSE frame, that node will pause transmission for the duration specified in the pause duration field. This register is not reset and must be initialized by the user.

Offset: 0x0EC 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	OPCODE																PAUSE_DUR															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 586. Opcode/Pause Duration Register (OPD)

Table 520. OPD field descriptions

Field	Description
OPCODE	Opcode field used in PAUSE frames. These bits are a constant, 0x0001.
PAUSE_DUR	Pause Duration field used in PAUSE frames.

30.5.4.14 Descriptor Individual Upper Address Register (IAUR)

The IAUR register is written by the user. This register contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

Offset: 0x118 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	IADDR1																															
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 587. Descriptor Individual Upper Address Register (IAUR)

Table 521. IAUR field descriptions

Field	Descriptions
IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

30.5.4.15 Descriptor Individual Lower Address Register (IALR)

The IALR register is written by the user. This register contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

Offset: 0x11C 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 W	IADDR2							
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 588. Descriptor Individual Lower Address Register (IALR)

Table 522. IALR field descriptions

Field	Description
IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0.

30.5.4.16 Descriptor Group Upper Address (GAUR)

The GAUR register is written by the user. This register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

Offset: 0x120 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 W	GADDR1							
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 589. Descriptor Group Upper Address Register (GAUR)

Table 523. GAUR field descriptions

Field	Description
GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

30.5.4.17 Descriptor Group Lower Address (GALR)

The GALR register is written by the user. This register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

Offset: 0x124 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 W	GADDR2							
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 590. Descriptor Group Lower Address Register (GALR)

Table 524. GALR field descriptions

Field	Description
GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

30.5.4.18 FIFO Transmit FIFO Watermark Register (TFWR)

The TFWR register contains a 2-bit field programmed by the user to control the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows the user to minimize transmit latency (TFWR = 0x) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value will minimize the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

Offset: 0x144 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	0	0	X_WMRK	
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

Figure 591. Transmit FIFO Watermark Register (TFWR)

Table 525. TFWR field descriptions

Field	Descriptions
X_WMRK	Number of bytes written to transmit FIFO before transmission of a frame begins 0x 64 bytes written 10 128 bytes written 11 192 bytes written

30.5.4.19 FIFO Receive Bound Register (FRBR)

The FRBR register is a eight bit register that the user can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR register to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Offset: 0x14C Access: User read

	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	1	R_BOUND								0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 592. FIFO Receive Bound Register (FRBR)

	31 … 16
Field	—
Reset	0000_0000_0000_0000
R/W	Read Only

	15 … 10	9 … 2	1 … 0
Field	—	R_BOUND	—
Reset	0000_0110_0000_0000		
R/W	Read Only		
Address	IPSBAR + 0x114C		

Figure 593. FIFO Receive Bound Register (FRBR)

Table 526. FRBR field descriptions

Name	Descriptions
R_BOUND	Highest valid FIFO RAM address.

30.5.4.20 FIFO Receive Start Register (FRSR)

The FRSR register is a eight bit register programmed by the user to indicate the starting address of the receive FIFO. FRSR marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to FRSR-4. The receive FIFO uses addresses from FRSR to FRBR inclusive.

The FRSR register is initialized by hardware at reset. FRSR only needs to be written to change the default value.

Offset: 0x150 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	1	R_FSTART								0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Figure 594. FIFO Receive Start Register (FRSR)

Table 527. FRSR field descriptions

Field	Descriptions
R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs.

30.5.4.21 Receive Descriptor Ring Start Register (ERDSR)

The ERDSR register is written by the user.

This register provides a pointer to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized by the user prior to operation.

Offset: 0x180 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	R_DES_START								0	0
W										
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— —	—	—

Figure 595. Receive Descriptor Ring Start Register (ERDSR)

Table 528. ERDSR field descriptions

Field	Descriptions
R_DES_START	Pointer to start of receive buffer descriptor queue.

30.5.4.22 Transmit Buffer Descriptor Ring Start (ETSDR)

The ETSDR register is written by the user.

This register provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). Bits 1 and 0 should be written to 0 by the user. Non-zero values in these two bit positions are ignored by the hardware.

This register is not reset and must be initialized by the user prior to operation.

Offset: 0x184 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	X_DES_START								0	0
W										
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— —	—	—

Figure 596. Transmit Buffer Descriptor Ring Start Register (ETDSR)

Table 529. ETDSR field descriptions

Field	Descriptions
X_DES_START	Pointer to start of transmit buffer descriptor queue.

30.5.4.23 Receive Buffer Size Register (EMRBR)

The EMRBR register is a 9-bit register programmed by the user. The EMRBR register dictates the maximum size of all receive buffers. Note that because receive frames will be truncated at 2k-1 bytes, only bits 10–4 are used. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR must be set to RCR[MAX_FL] or larger. The EMRBR must be evenly divisible by 16. To insure this, bits 3-0 are forced low. To minimize bus utilization (descriptor fetches) it is recommended that EMRBR be greater than or equal to 256 bytes.

The EMRBR register does not reset, and must be initialized by the user.

Offset: 0x188 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	R_BUF_SIZE							0	0	0	0
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 597. Receive Buffer Size Register (EMRBR)

Table 530. EMRBR field descriptions

Field	Descriptions
R_BUF_SIZE	Receive buffer size.

30.6 Buffer Descriptors

This section provides a description of the operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields.

30.6.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames must reside in memory external to the FEC. The data for a frame is placed in one or more buffers. Associated with each buffer is a buffer descriptor (BD) which contains a starting address (pointer), data length, and status/control information (which contains the current state for the buffer). To permit maximum user flexibility, the BDs are also located in external memory and are read in by the FEC DMA engine.

Software “produces” buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD[E] or TxBD[R] bit “produces” the buffer. Software writing to either the TDAR or RDAR tells the FEC that a buffer has been placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and “consumes” the buffers after they have been produced. After the data DMA is complete and the buffer descriptor status bits have been written by the DMA engine, the RxBD[E] or TxBD[R] bit will be cleared by hardware to signal the buffer has been “consumed.” Software may poll the BDs to detect when the buffers have been consumed or may rely on the buffer/frame interrupts. These buffers may then be processed by the driver and returned to the free list.

The ECR[ETHER_EN] signal operates as a reset to the BD/DMA logic. When ECR[ETHER_EN] is deasserted the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before the ECR[ETHER_EN] bit is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR defines the starting address for transmit BDs. The last buffer descriptor in each ring is defined by the Wrap (W) bit. When set, W indicates that the next descriptor in the ring is at the location pointed to by ERDSR and ETDSR for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

30.6.1.1 Driver/DMA Operation with Transmit BDs

Typically a transmit frame will be divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a 2nd buffer, IP header in a 3rd buffer, Ethernet/IEEE 802.3 header in a 4th buffer. The Ethernet MAC does not prepend the Ethernet header (Destination Address, Source Address, Length/Type field(s)), so this must be provided by the driver in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. Whether the CRC is appended by the MAC or by the driver is determined by the TC bit in the transmit BD which must be set by the driver.

The driver (TxBD software producer) should set up Tx BDs in such a way that a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length and control (W, L, TC, ABC) and then the TxBD[R] bits should be set = 1 in reverse order (3rd, 2nd, 1st BD) to insure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA Controller could DMA the first BD before the 2nd was made available, potentially causing a transmit FIFO underrun.

In the FEC, the DMA is notified by the driver that new transmit frame(s) are available by writing to the TDAR register. When this register is written to (data value is not significant) the FEC RISC will tell the DMA to read the next transmit BD in the ring. Once started, the RISC + DMA will continue to read and interpret transmit BDs in order and DMA the associated buffers, until a transmit BD is encountered with the R bit = 0. At this point the FEC will poll this BD one more time. If the R bit = 0 the second time, then the RISC will stop the transmit descriptor read process until software sets up another transmit frame and writes to TDAR.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

30.6.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR register.

The driver (RxBD software producer) should set up some number of “empty” buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) will consume these buffers by filling them with data as frames are received and clearing the E bit and writing to the L (1 indicates last buffer in frame) bit, the frame status bits (if L = 1) and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD will be written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end of frame buffers the receive BD will be written with L = 1 and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer will be written with the length of the entire frame, not just the length of the last buffer.

For simplicity the driver may assign the default receive buffer length to be large enough to contain an entire frame, keeping in mind that a malfunction on the network or out of spec implementation could result in giant frames. Frames of 2k (2048) bytes or larger are truncated by the FEC at 2047 bytes so software is guaranteed never to see a receive frame larger than 2047 bytes.

Similar to transmit, the FEC will poll the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR register. As frames are received the FEC will fill receive buffers and update the associated BDs, then read the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit = 0, it will poll this BD once more. If the BD = 0 a second time the FEC will stop reading receive BDs until the driver writes to RDAR.

30.6.2 Ethernet Receive Buffer Descriptor (RxBD)

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in second longword. When the buffer has been DMA'd, the Ethernet controller will modify the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and write the length of the used portion of the buffer in the first longword. The M, BC, MC, LG, NO, CR, OV and TR bits in the first longword of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

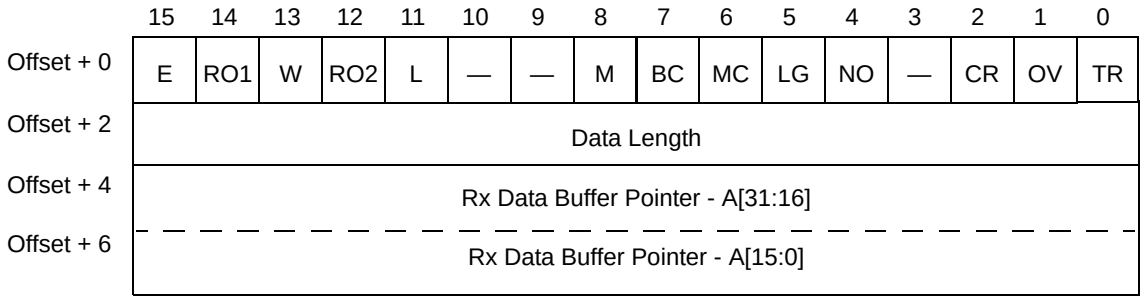


Figure 598. Receive Buffer Descriptor (RxBD)

Table 531. Receive Buffer Descriptor Field Definitions

Word	Location	Field Name	Description
Offset + 0	Bit 15	E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD has been filled with received data, or data reception has been aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	Bit 14	RO1	Receive software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	Bit 13	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ERDSR.
Offset + 0	Bit 12	RO2	Receive software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	Bit 11	L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.

Table 531. Receive Buffer Descriptor Field Definitions (continued)

Word	Location	Field Name	Description
Offset + 0	Bits 10–9	—	Reserved.
Offset + 0	Bit 8	M	Miss. Written by the FEC. This bit is set by the FEC for frames that were accepted in promiscuous mode, but were flagged as a “miss” by the internal address recognition. Thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
Offset + 0	Bit 7	BC	Will be set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	Bit 6	MC	Will be set if the DA is multicast and not BC.
Offset + 0	Bit 5	LG	Rx frame length violation. Written by the FEC. A frame length greater than RCR[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	Bit 4	NO	Receive non-octet aligned frame. Written by the FEC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L-bit is set. If this bit is set the CR bit will not be set.
Offset + 0	Bit 3	--	Reserved.
Offset + 0	Bit 2	CR	Receive CRC error. Written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L-bit is set.
Offset + 0	Bit 1	OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and will be zero. This bit is valid only if the L-bit is set.
Offset + 0	Bit 0	TR	Will be set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set the frame should be discarded and the other error bits should be ignored as they may be incorrect.
Offset + 2	Bits [15:0]	Data Length	Data length. Written by the FEC. Data length is the number of octets written by the FEC into this BD’s data buffer if L = 0 (the value will be equal to EMRBR), or the length of the frame including CRC if L = 1. It is written by the FEC once as the BD is closed.
0ffset + 4	Bits [15:0]	A[31:16]	RX data buffer pointer, bits [31:16]
Offset + 6	Bits [15:0]	A[15:0]	RX data buffer pointer, bits [15:0]

Note: Whenever the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to RDAR.

30.6.3 Ethernet Transmit Buffer Descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel’s TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (R

bit) when DMA of the buffer is complete. In the TxBD the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword, and the buffer pointer in the second longword.

The FEC will set the R bit = 0 in the first longword of the BD when the buffer has been DMA'd. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See *Section 30.5.3: MIB Block Counters Memory Map*, for more details.

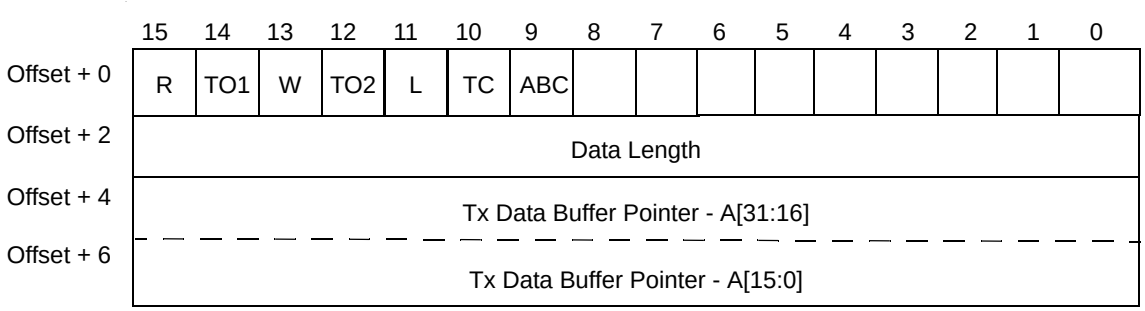


Figure 599. Transmit Buffer Descriptor (TxBD)

Table 532. Transmit Buffer Descriptor Field Definitions

Word	Location	Field Name	Description
Offset + 0	Bit 15	R	Ready. Written by the FEC and the user. 0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD may be written by the user once this bit is set.
Offset + 0	Bit 14	TO1	Transmit software ownership. This field is reserved for software use. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	Bit 13	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ETDSR.
Offset + 0	BIt 12	TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	Bit 11	L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.
Offset + 0	Bit 10	TC	Tx CRC. Written by user (only valid if L = 1). 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.

Table 532. Transmit Buffer Descriptor Field Definitions (continued)

Word	Location	Field Name	Description
Offset + 0	Bit 9	ABC	Append bad CRC. Written by user (only valid if L = 1). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value).
Offset + 0	Bits [8:0]	—	Reserved.
Offset + 2	Bits [15:0]	Data Length	Data Length, written by user. Data length is the number of octets the FEC should transmit from this BD's data buffer. It is never modified by the FEC. Bits [15:5] are used by the DMA engine, bits[4:0] are ignored.
Offset + 4	Bits [15:0]	A[31:16]	Tx data buffer pointer, bits [31:16](1)
Offset + 6	Bits [15:0]	A[15:0]	Tx data buffer pointer, bits [15:0].

1. The transmit buffer pointer, which contains the address of the associated data buffer, may be even or odd. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

Note: Once the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame should be to set the R bit in the first BD for the frame. The driver should follow that with a write to TDAR which will trigger the FEC to poll the next BD in the ring.

31 Timers

This chapter describes the timer modules implemented on the microcontroller:

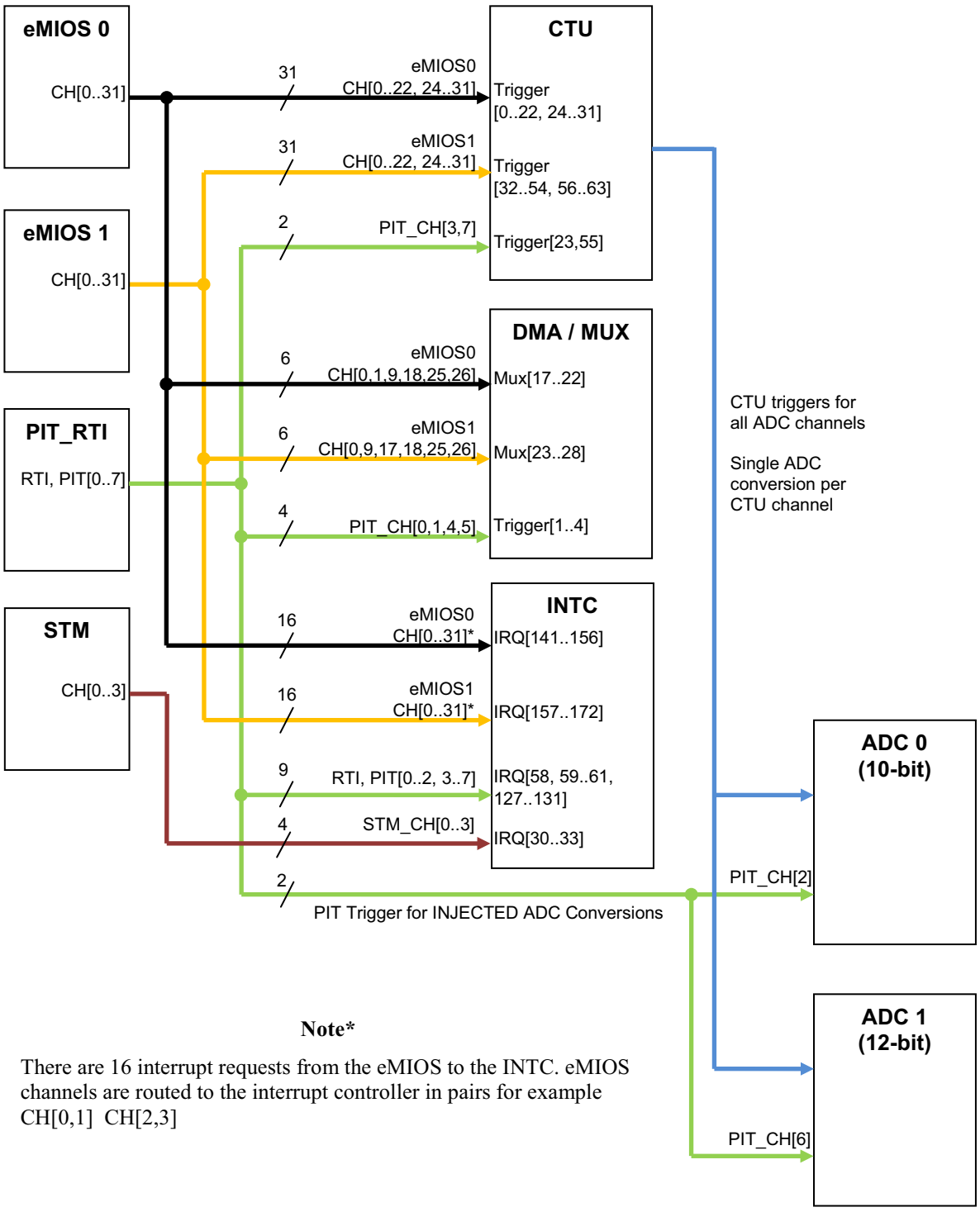
- *Section 31.2: System Timer Module (STM)*
- *Section 31.3: Enhanced Modular IO Subsystem (eMIOS)*
- *Section 31.4: Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI)*

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.

31.1 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 600 details the interaction between the timers and the eDMA, INTC, CTU, and ADC.



Note*

There are 16 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 600. Interaction between timers and relevant peripherals

31.1.1 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 analyze 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.

31.1.2 Overview of the eMIOS

There are two 32-channel eMIOS modules. 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 31.3: 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.

On eMIOS_0, PWM signals can be automatically serialised over the DSPI. This not only saves on pins but allows direct connection to a SPI based external lamp driver. Once configured, the eMIOS serialisation feature works with no or little CPU intervention and DSPI packets are transmitted automatically, triggered by eMIOS PWM flags.

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 31.3.4.1.1.12: 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 out with 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 533* and *Table 534* show the eMIOS channel numbers that are available on each pin. The color shading matches the channel configuration diagram in the eMIOS section.

Table 533. eMIOS_0 channel to pin mapping

Channel	Pin function				Channel	Pin function		
	ATL1	ALT2	ALT3			ATL1	ALT2	ALT3
UC[0]	PA[0]		PA[14]		UC[16]	PE[0]		
UC[1]	PA[1]		PA[15]		UC[17]	PE[1]		
UC[2]	PA[2]				UC[18]	PE[2]		
UC[3]	PB[11]	PC[8]			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]	PC[9]			UC[23]	PE[7], PF[6]	PE[9]	
UC[8]	PA[8]				UC[24]	PE[11], 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[28]	PI[0]	PA[12]	
UC[13]	PC[13], PF[3]		PA[0]		UC[29]	PI[1]	PA[13]	
UC[14]	PC[14], PF[4]	PA[8]			UC[30]	PI[2]	PB[0]	PB[2]
UC[15]	PC[15]				UC[31]	PB[3], PI[3]	PB[1]	

Table 534. eMIOS_1 channel to pin mapping

Channel	Pin function				Channel	Pin function		
	ATL1	ALT2	ALT3			ATL1	ALT2	ALT3
UC[0]	PG[14], PK[3]				UC[16]	PG[7]		
UC[1]	PG[15]		PK[4]		UC[17]	PG[8]		PH[15]
UC[2]	PH[0]		PF[10]		UC[18]	PG[9]	PJ[4]	
UC[3]	PH[1]	PF[11]			UC[19]		PE[12]	
UC[4]	PF[15], 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]		PH[12]
UC[10]	PH[8]				UC[26]	PF[13]		PH[13]
UC[11]	PG[2]				UC[27]		PF[14]	PH[14]

Table 534. eMIOS_1 channel to pin mapping (continued)

Channel	Pin function				Channel	Pin function		
	ATL1	ALT2	ALT3			ATL1	ALT2	ALT3
UC[12]	PG[3]				UC[28]	PI[4]	PC[6]	
UC[13]	PG[4]				UC[29]	PI[5]	PC[7]	
UC[14]	PG[5]				UC[30]	PI[6]	PG[7]	PE[10]
UC[15]	PG[6]				UC[31]	PC[4], PI[7]	PG[10]	

31.1.3 Overview of the PIT_RTI

The PIT_RTI module consists of:

- 8 Periodic Interrupt Timers (PITs) clocked from the system clock
- 1 Real Time Interrupt (RTI) block clocked from the FXOSC so that it can be used for system wakeup

Out of reset, the PIT and RTI timers are disabled. There is a global disable control bit for all of the PIT timers and a separate independent disable for the RTI. Before using the timers, software must clear the appropriate disabled bit. Each of the PIT timers and the RTI are effectively standalone entities and each have their own timer and control registers.

The PIT_RTI 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:

- 4 of the PIT channels can be used as an eDMA trigger
- 2 PIT channel can be used to trigger a CTU ADC conversion (single)
- 2 PIT channels (1 per ADC module) 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 PIT and RTI are both available in all modes where a system clock is generated. The RTI is also available in stop mode as a wakeup source as long as the external FXOSC is enabled.

There are no external pins associated with the PIT_RTI.

31.2 System Timer Module (STM)

31.2.1 Introduction

31.2.1.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

31.2.1.2 Features

The STM has the following features:

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

31.2.1.3 Modes of operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR register. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

31.2.2 External signal description

The STM does not have any external interface signals.

31.2.3 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.

31.2.3.1 Memory map

The STM memory map is shown in *Table 535*.

Table 535. STM memory map

Base address: 0xFFF3_C000		
Address offset	**Register**	**Location**
0x0000	STM Control Register (STM_CR)	*on page 1012*
0x0004	STM Counter Value (STM_CNT)	*on page 1013*
0x0008–0x000C	Reserved	
0x0010	STM Channel 0 Control Register (STM_CCR0)	*on page 1013*
0x0014	STM Channel 0 Interrupt Register (STM_CIR0)	*on page 1013*
0x0018	STM Channel 0 Compare Register (STM_CMP0)	*on page 1014*
0x001C	Reserved	
0x0020	STM Channel 1 Control Register (STM_CCR1)	*on page 1013*
0x0024	STM Channel 1 Interrupt Register (STM_CIR1)	*on page 1013*
0x0028	STM Channel 1 Compare Register (STM_CMP1)	*on page 1014*
0x002C	Reserved	
0x0030	STM Channel 2 Control Register (STM_CCR2)	*on page 1013*
0x0034	STM Channel 2 Interrupt Register (STM_CIR2)	*on page 1013*

Table 535. STM memory map (continued)

Base address: 0xFFF3_C000		
Address offset	**Register**	**Location**
0x0038	STM Channel 2 Compare Register (STM_CMP2)	*on page 1014*
0x003C	Reserved	
0x0040	STM Channel 3 Control Register (STM_CCR3)	*on page 1013*
0x0044	STM Channel 3 Interrupt Register (STM_CIR3)	*on page 1013*
0x0048	STM Channel 3 Compare Register (STM_CMP3)	*on page 1014*
0x004C–0x3FFF	Reserved	

31.2.3.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

31.2.3.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

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

Figure 601. STM Control Register (STM_CR)

Table 536. 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.

31.2.3.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

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 W	CNT							
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

Figure 602. STM Count Register (STM_CNT)

Table 537. 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.

31.2.3.2.3 STM Channel Control Register (STM_CCRn)

The STM Channel Control Register (STM_CCRn) has the enable bit for channel n of the timer.

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

Figure 603. STM Channel Control Register (STM_CCRn)

Table 538. STM_CCRn field descriptions

Field	Description
CEN	Channel Enable. 0 = The channel is disabled. 1 = The channel is enabled.

31.2.3.2.4 STM Channel Interrupt Register (STM_CIRn)

The STM Channel Interrupt Register (STM_CIRn) has the interrupt flag for channel n of the timer.

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

Figure 604. STM Channel Interrupt Register (STM_CIRn)

Table 539. STM_CIRn field descriptions

Field	Description
CIF	Channel Interrupt Flag 0 = No interrupt request. 1 = Interrupt request due to a match on the channel.

31.2.3.2.5 STM Channel Compare Register (STM_CMPn)

The STM channel compare register (STM_CMPn) holds the compare value for channel n.

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

Figure 605. STM Channel Compare Register (STM_CMPn)

Table 540. 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.

31.2.4 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.

31.3 Enhanced Modular IO Subsystem (eMIOS)

31.3.1 Introduction

31.3.1.1 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.

31.3.1.2 Features of the eMIOS module

- 2 eMIOS blocks with 32 channels each
  - All64 channels with OPWMT, which can be connected to the CTU
  - 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
- Flag outputs of channels 8-11 of both eMIOSs are used to disable the outputs of other channels. They form the ODIS bits.
- State of the UC can be frozen for debug purposes
- Motor control capability

31.3.1.3 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

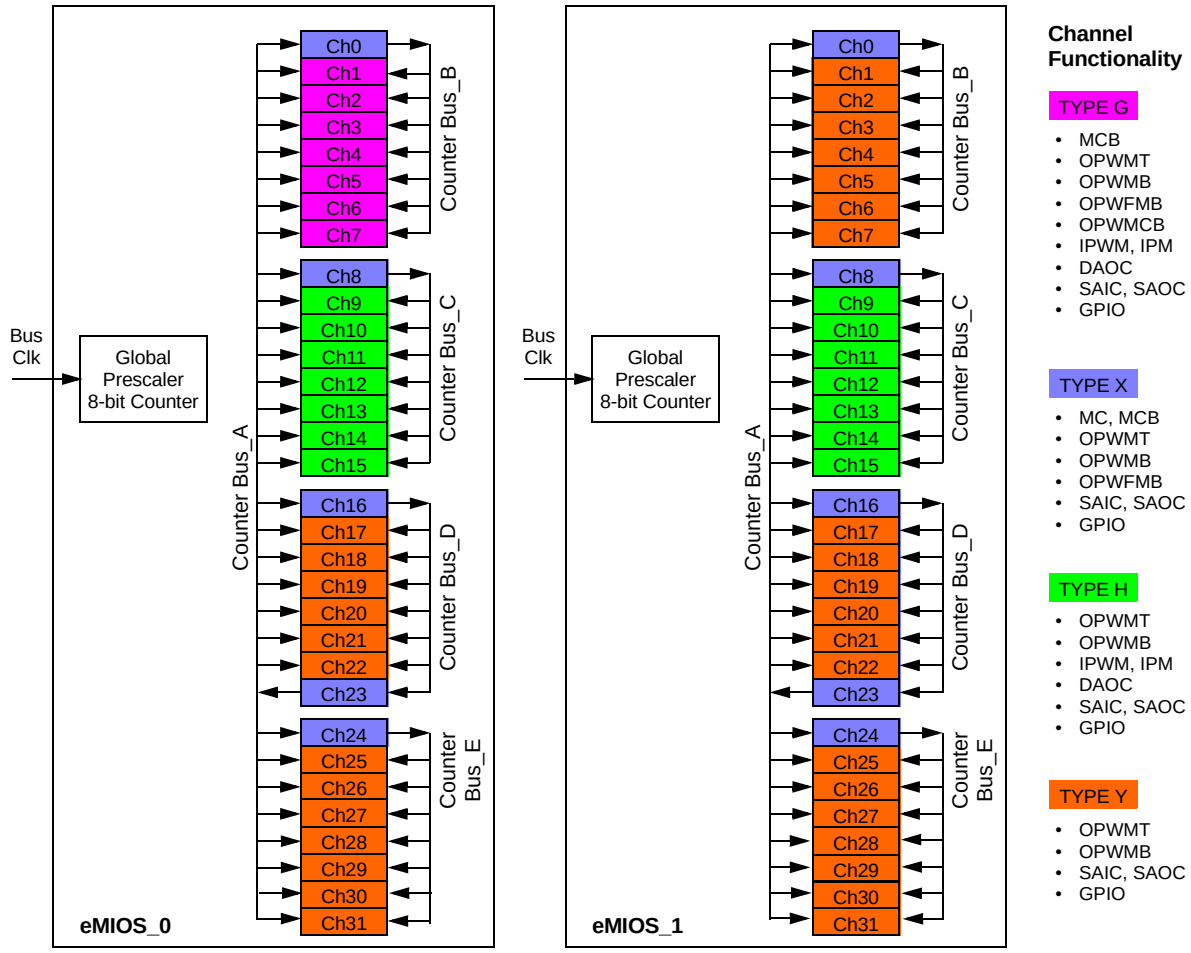
These modes are described in *Section 31.3.4.1.1: UC modes of operation*.

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 31.3.3.2.8: eMIOS UC Control Register (EMIOSC[n])*.

31.3.1.4 Channel implementation

Figure 606 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 606. Channel configuration

31.3.1.4.1 Channel mode selection

Channel modes are selected using the mode selection bits MODE[0:6] in the *Section 31.3.3.2.8: eMIOS UC Control Register (EMIOSC[n])*. *Table 553* provides the specific mode selection settings for the eMIOS implementation on this device.

31.3.2 External signal description

For information on eMIOS external signals on this device, please refer to the signal description chapter of the reference manual.

31.3.3 Memory map and register description

31.3.3.1 Memory maps

The overall address map organization is shown in *Table 541*.

31.3.3.1.1 Unified Channel memory map

Table 541. 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 1019*
0x004–0x007	*eMIOS Global FLAG (EMIOSGFLAG) Register*	*on page 1020*
0x008–0x00B	*eMIOS Output Update Disable (EMIOSOUDIS) Register*	*on page 1021*
0x00C–0x00F	*eMIOS Disable Channel (EMIOSUCDIS) Register*	*on page 1022*
0x010–0x01F	Reserved	—
0x020–0x11F	Channel [0] to Channel [7]	—
0x120–0x21F	Channel [8] to Channel [15]	—
0x220–0x31F	Channel [16] to Channel [23]	—
0x320–0x41F	Channel [24] to Channel [31]	
0x420–0xFFF	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 542 describes the Unified Channel memory map.

Table 542. Unified Channel memory map

UC[n] base address	Description	Location
0x00	*eMIOS UC A Register (EMIOSA[n])*	*on page 1022*
0x04	*eMIOS UC B Register (EMIOSB[n])*	*on page 1023*
0x08	*eMIOS UC Counter Register (EMIOSCNT[n])*	*on page 1024*
0x0C	*eMIOS UC Control Register (EMIOSC[n])*	*on page 1024*
0x10	*eMIOS UC Status Register (EMIOSS[n])*	*on page 1028*
0x14	*eMIOS UC Alternate A Register (EMIOSALTA[n])*	*on page 1029*
0x18–0x1F	Reserved	—

31.3.3.2 Register description

All control registers are 32 bits wide. Data registers and counter registers are 16 bits wide.

31.3.3.2.1 eMIOS Module Configuration Register (EMIOSMCR)

The EMIOSMCR contains global control bits for the eMIOS block.

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

Figure 607. eMIOS Module Configuration Register (EMIOSMCR)

Table 543. 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 543. 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 544*.

Table 544. Global prescaler clock divider

GPRE	Divide ratio
00000000	1
00000001	2
00000010	3
00000011	4
. . . .	. . . .
11111110	255
11111111	256

31.3.3.2.2 eMIOS Global FLAG (EMIOSGFLAG) Register

The EMIOSGFLAG is a read-only register that groups the flag bits (F[31:0]) 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.

Address: eMIOS base address +0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F31	F30	F29	F28	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

Figure 608. eMIOS Global FLAG (EMIOSGFLAG) Register

Table 545. EMIOSGFLAG field descriptions

Field	Description
Fn	Channel [n] Flag bit

31.3.3.2.3 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	OU31	OU30	OU29	OU28	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

Figure 609. eMIOS Output Update Disable (EMIOSOUDIS) Register

Table 546. 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.

31.3.3.2.4 eMIOS Disable 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 W	CHDIS31	CHDIS30	CHDIS29	CHDIS28	CHDIS27	CHDIS26	CHDIS25	CHDIS24	CHDIS23	CHDIS22	CHDIS21	CHDIS20	CHDIS19	CHDIS18	CHDIS17	CHDIS16
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 W	CHDIS15	CHDIS14	CHDIS13	CHDIS12	CHDIS11	CHDIS10	CHDIS9	CHDIS8	CHDIS7	CHDIS6	CHDIS5	CHDIS4	CHDIS3	CHDIS2	CHDIS1	CHDIS0
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

Figure 610. eMIOS Enable Channel (EMIOSUCDIS) Register

Table 547. 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

31.3.3.2.5 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 W	A															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 611. eMIOS UC A Register (EMIOSA[n])

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 548* summarizes the EMIOSA[n] writing and reading accesses for all operation modes. For more information see *Section 31.3.4.1.1: UC modes of operation*.

31.3.3.2.6 eMIOS UC B Register (EMIOSB[n])

Address: UC[n] base address + 0x04

	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	B															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 612. eMIOS UC B Register (EMIOSB[n])

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 548* summarizes the EMIOSB[n] writing and reading accesses for all operation modes. For more information see *Section 31.3.4.1.1: 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 548. 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.

31.3.3.2.7 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															
W1																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 613. eMIOS UC Counter Register (EMIOSCNT[n])

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 31.3.4.1.1: 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.

31.3.3.2.8 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.

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

Figure 614. eMIOS UC Control Register (EMIOSC[n])

Table 549. 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 550*.
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 request, as a DMA request or as a CTU trigger. The choice between a DMA request or a CTU trigger is determined by the value of bit TM in the register CTU_EVTCFGRx (refer to the CTU chapter of the reference manual). 1 = Flag/overrun assigned toDMA request or 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 551*. 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
FEN	FLAG Enable bit The FEN bit allows the Unified Channel FLAG bit to generate an interrupt signal or a DMA request 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 DMA requestor a CTU trigger) 0 = Disable (FLAG does not generate an interrupt request or DMA requestor 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 552* for details.

Table 549. EMIOSC[n] field descriptions (continued)

Field	Description
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
MODE	Mode selection The MODE field selects the mode of operation of the Unified Channel, as shown in *Table 553*. **Note:** If a reserved value is written to mode the results are unpredictable.

Table 550. UC internalprescaler clock divider

UCPRE	Divide ratio
00	1
01	2
10	3
11	4

Table 551. 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 552. 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 553. 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 31.3.4.1.1: UC modes of operation* for details.

31.3.3.2.9 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

Figure 615. eMIOS UC Status Register (EMIOSS[n])

Table 554. 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 DMA controller or theCTU.

31.3.3.2.10 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

Figure 616. eMIOS UC Alternate A register (EMIOSALTA[n])

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 548* summarizes the EMIOSALTA[n] writing and reading accesses for all operation modes. Please, see *Section 31.3.4.1.1.1: General purpose Input/Output (GPIO) mode*, *Section 31.3.4.1.1.12: Output Pulse Width Modulation with Trigger (OPWMT) mode* for a more detailed description of the use of EMIOSALTA[n] register.

31.3.4 Functional description

The four types of channels of the eMIOS (types X, Y, G and H) can operate in the modes as listed in *Figure 606*. The eMIOS provides independently operating unified channels (UC) that can be configured and accessed by a host MCU. Up to three time bases[q] can be shared by the channels through five counter buses[r] and each unified channel can generate its own time base[s]. The eMIOS block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

q. Time bases can be supplied by:

a) channel 23 to all unified channels

b) channel 0 to channels 0 to 7, by channel 8 to channels 8 to 15, by channel 16 to channels 16 to 23, by channel 24 to channels 24 to 31

c) channel's internal counter when available.

r. Internal eMIOS architecture have one global counter bus A and four local counter buses B, C, D, and E, that distribute the time bases described in Note 1 (a) and (b).

s. 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.

31.3.4.1 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

31.3.4.1.1 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 614* 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.

31.3.4.1.1.1 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.

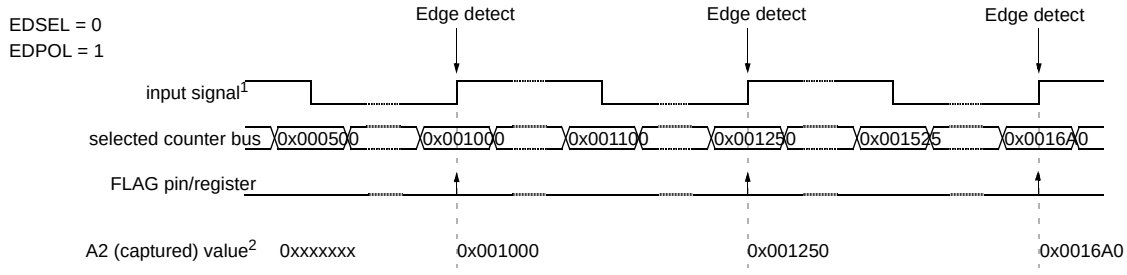
31.3.4.1.1.2 Single Action Input Capture (SAIC) mode

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.

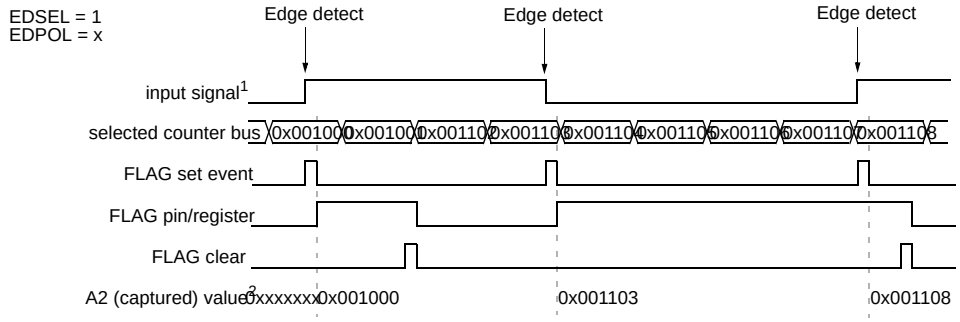
The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOSC[n] register.

Figure 617 and *Figure 618* show how the Unified Channel can be used for input capture.



Notes: 1. After input filter
2. EMIOSA[n] <= A2

Figure 617. Single action input capture with rising edge triggering example



Notes: 1. After input filter
2. EMIOSA[n] <= A2

Figure 618. Single action input capture with both edges triggering example

31.3.4.1.1.3 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.

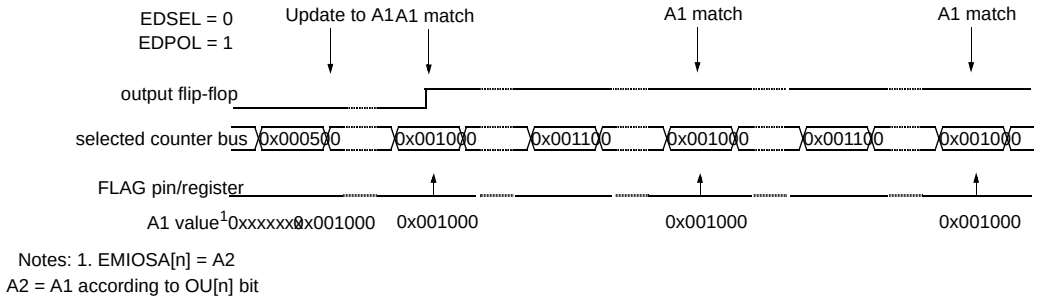
An output compare match can be simulated in software by setting the FORCMA bit in EMIOSC[n] register. In this case, the FLAG bit is not set.

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].

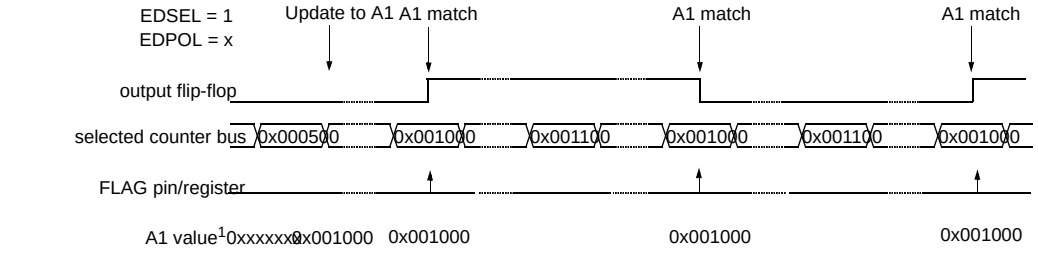
Figure 619 and *Figure 620* show how the Unified Channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode the matches are enabled thus the desired match 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 621*).

Note: The channel internal counter in SAOC mode is free-running. It starts counting as soon as the SAOC mode is entered.



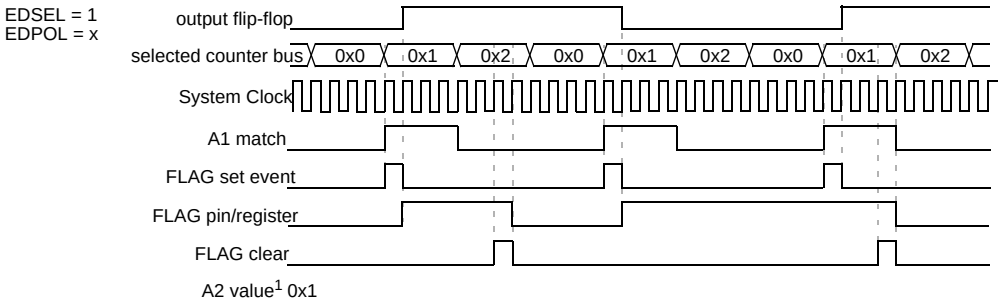
Notes: 1. EMIOSA[n] = A2
A2 = A1 according to OU[n] bit

Figure 619. SAOC example with EDPOL value being transferred to the output flip-flop



Notes: 1. EMIOSA[n] = A2
A2 = A1 according to OU[n] bit

Figure 620. SAOC example toggling the output flip-flop



Note: 1. EMIOSA[n] <= A2

Figure 621. SAOC example with flag behavior

31.3.4.1.1.4 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 622 shows how the Unified Channel can be used for input pulse width measurement.

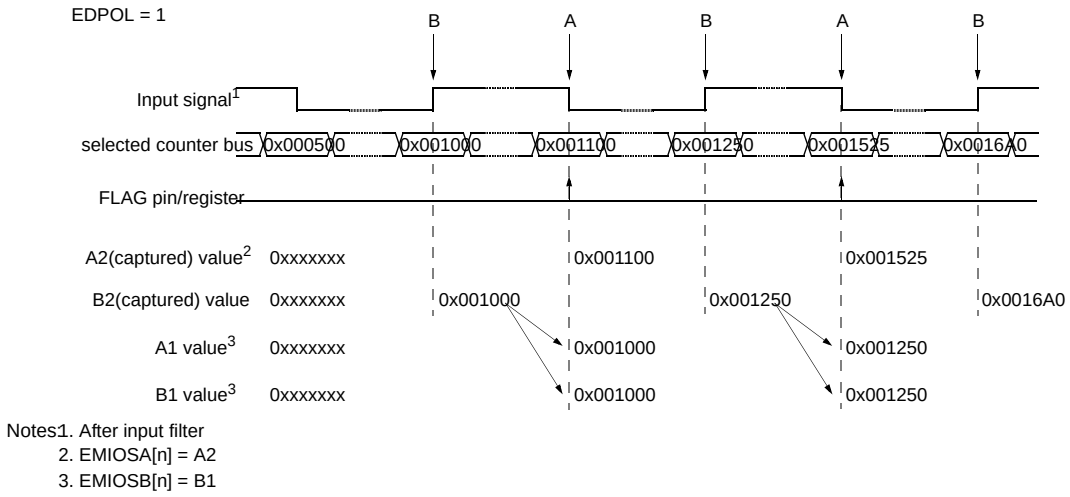


Figure 622. Input pulse width measurement example

Figure 623 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.

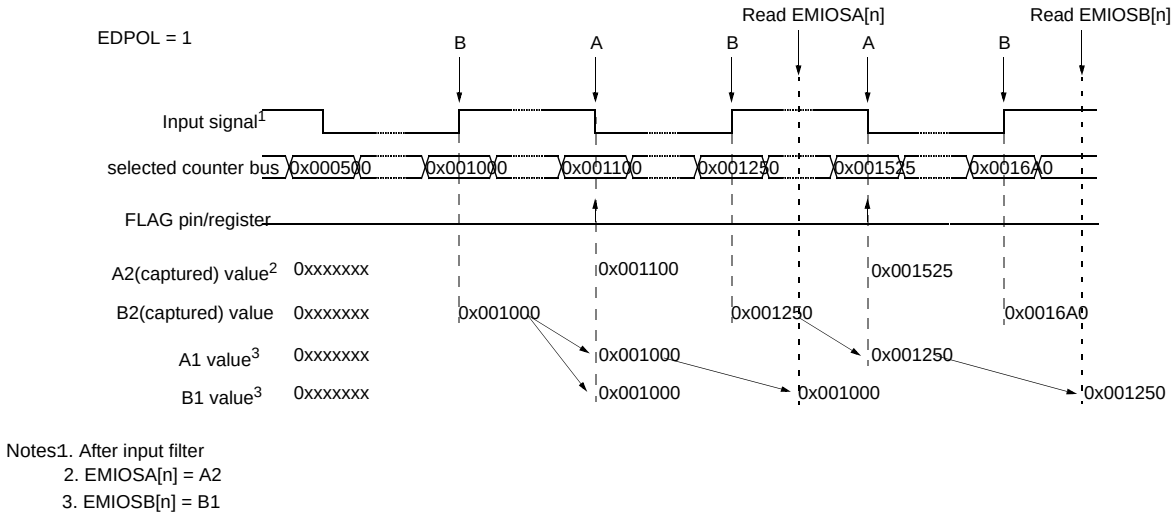


Figure 623. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads

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.

31.3.4.1.1.5 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.

Note: The input signal must have at least four system clock cycles period in order to be properly captured by the synchronization logic at the channel input even if the input filter is in by-pass mode.

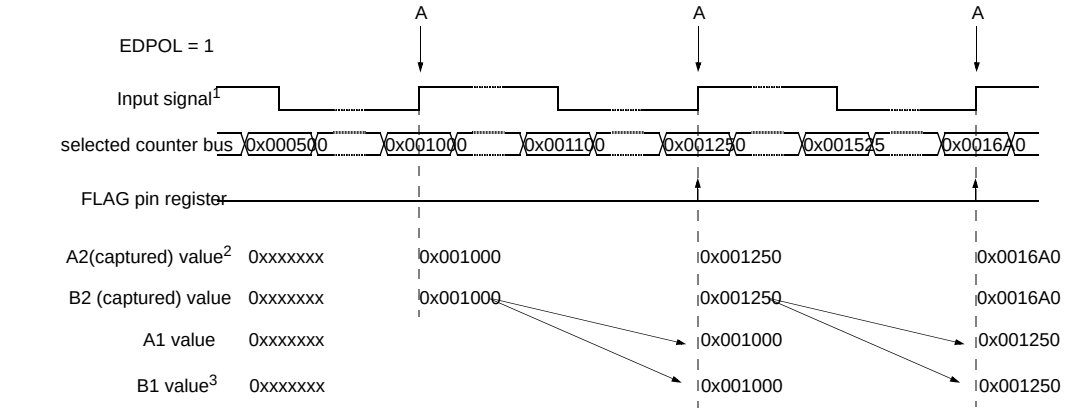
When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second 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 624 shows how the Unified Channel can be used for input period measurement.

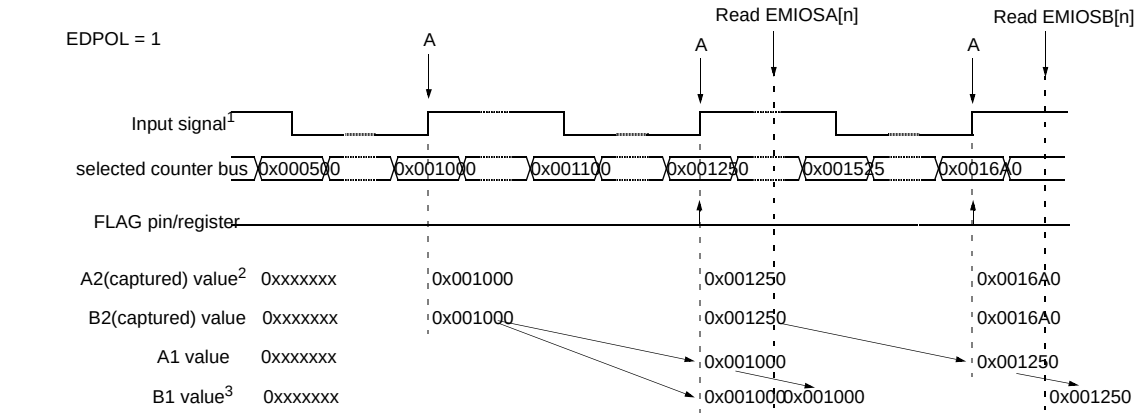


Notes: 1. After input filter
2. EMIOSA[n] = A2
3. EMIOSB[n] = B1

Figure 624. Input period measurement example

Figure 625 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 625* example.



Notes: 1. After input filter
2. EMIOSA[n] = A2
3. EMIOSB[n] = B1

Figure 625. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads

31.3.4.1.1.6 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 628*). 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.

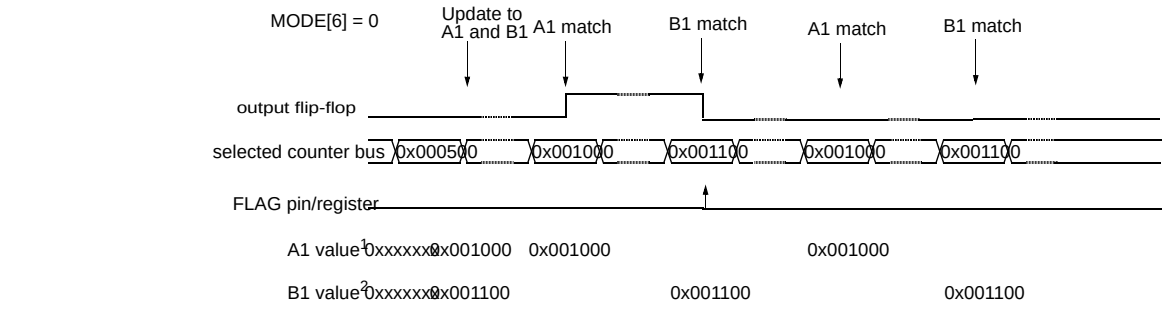
MODE[6] controls if the FLAG is set on both matches (MODE[0:6] = 0000111) or just on the B match (MODE[0:6] = 0000110). FLAG bit assertion depends on comparator enabling.

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.

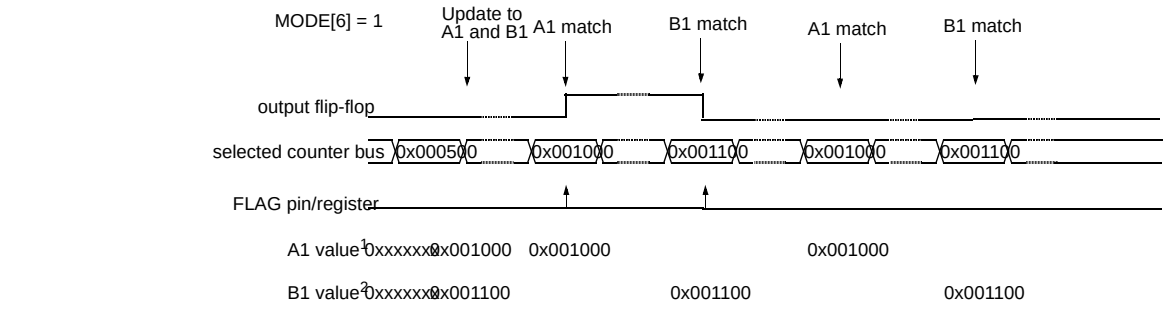
Figure 626 and *Figure 627* show how the Unified Channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



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

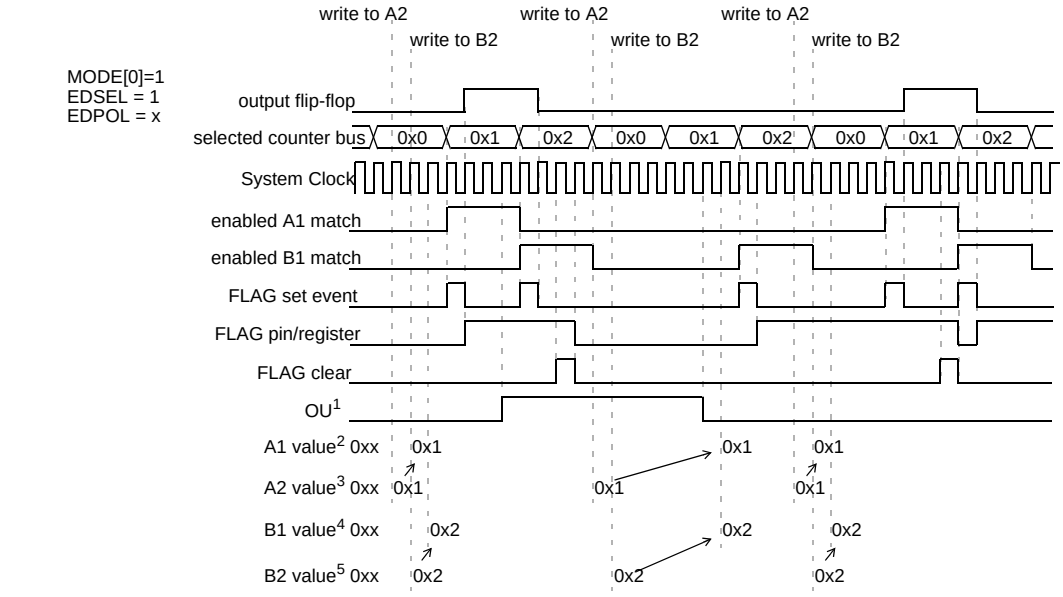
Figure 626. Double action output compare with FLAG set on the second match



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 627. 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 628. DAOC with transfer disabling example

31.3.4.1.1.7 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 651* and *Figure 652*.
  - 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 651* and *Figure 653*.

- 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 651* and *Figure 654*.
  - 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 651* and *Figure 654*.

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.

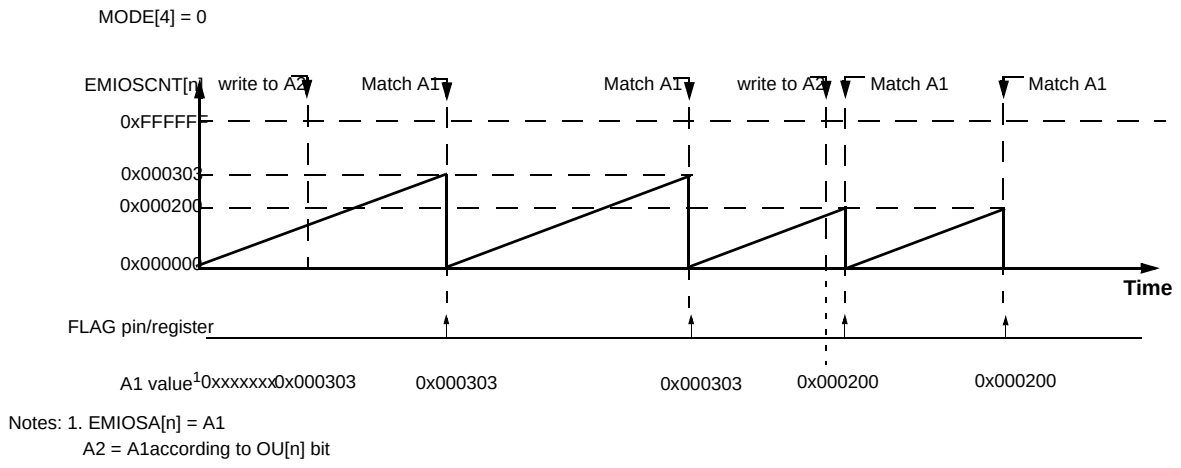
When in up/down count mode (MODE[0:6] = 00101bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the 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.

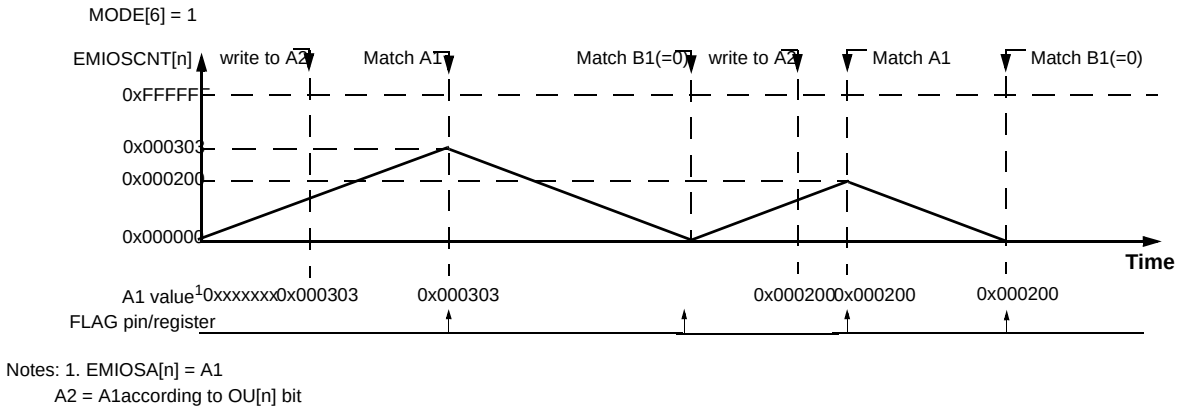
Register B2 has no effect in MC mode. Nevertheless, register B2 can be accessed for reads and writes by addressing EMIOSB.

Figure 629 and *Figure 630* show how the Unified Channel can be used as modulus counter in up mode and up/down mode, respectively.



Notes: 1. EMIOSA[n] = A1
A2 = A1according to OU[n] bit

Figure 629. Modulus Counter Up mode example



Notes: 1. EMIOSA[n] = A1
A2 = A1according to OU[n] bit

Figure 630. Modulus Counter Up/Down mode example

31.3.4.1.1.8 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 631 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.

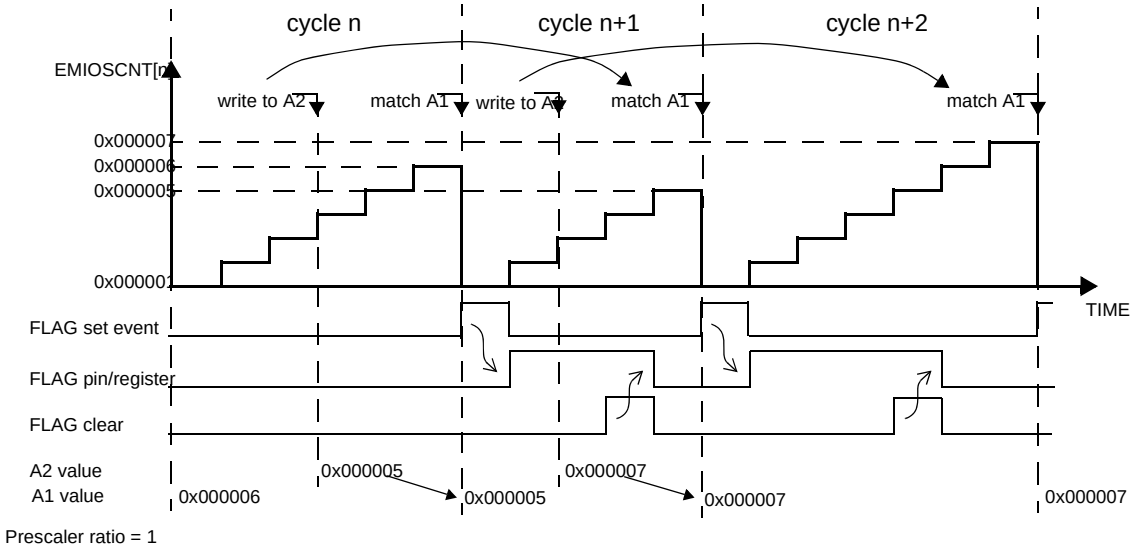


Figure 631. Modulus Counter Buffered (MCB) Up Count mode

Figure 632 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.

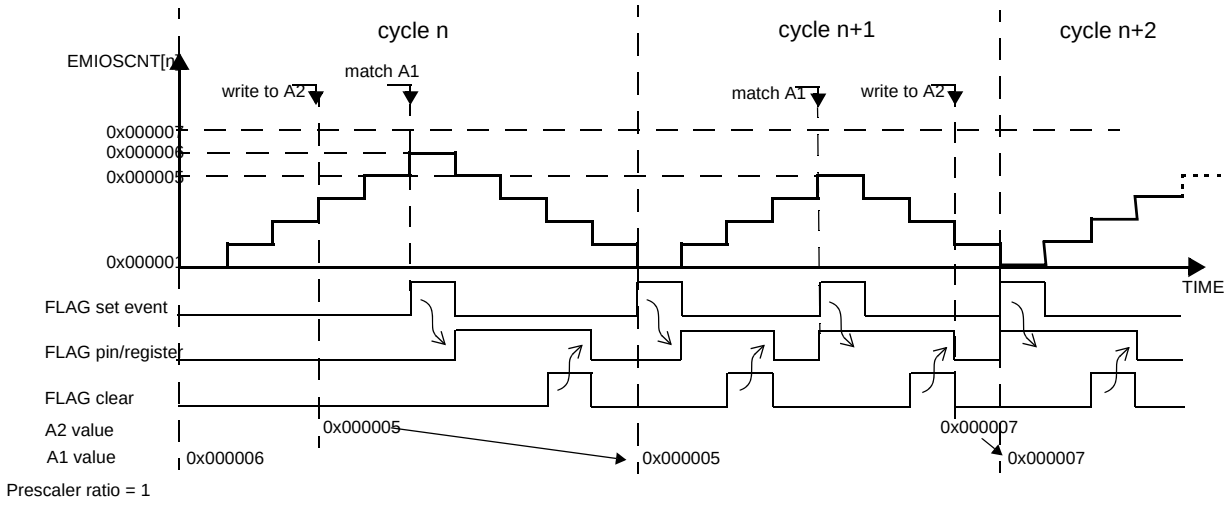


Figure 632. Modulus Counter Buffered (MCB) Up/Down mode

Figure 633 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.

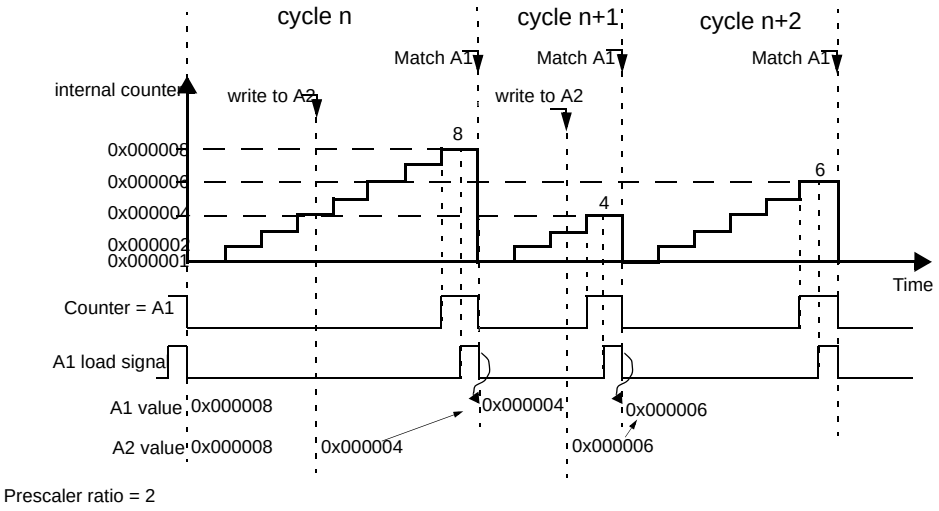


Figure 633. MCB Mode A1 Register Update in Up Counter mode

Figure 634 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.

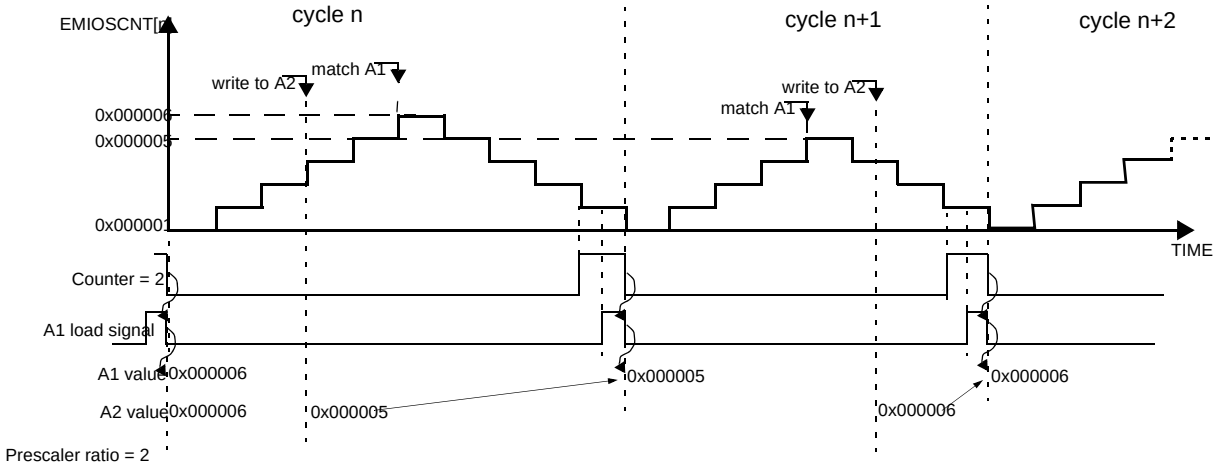


Figure 634. MCB Mode A1 Register Update in Up/Down Counter mode

31.3.4.1.1.9 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 635 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 635* the internal counter prescaler has a ratio of two.

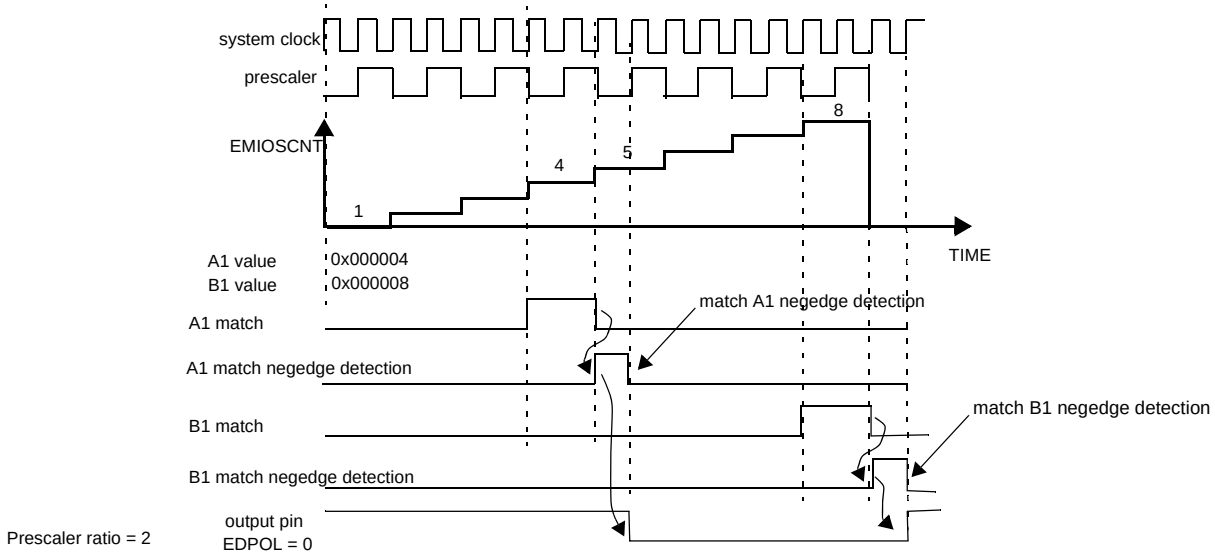


Figure 635. OPWFMB A1 and B1 match to Output Register Delay

Figure 636 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.

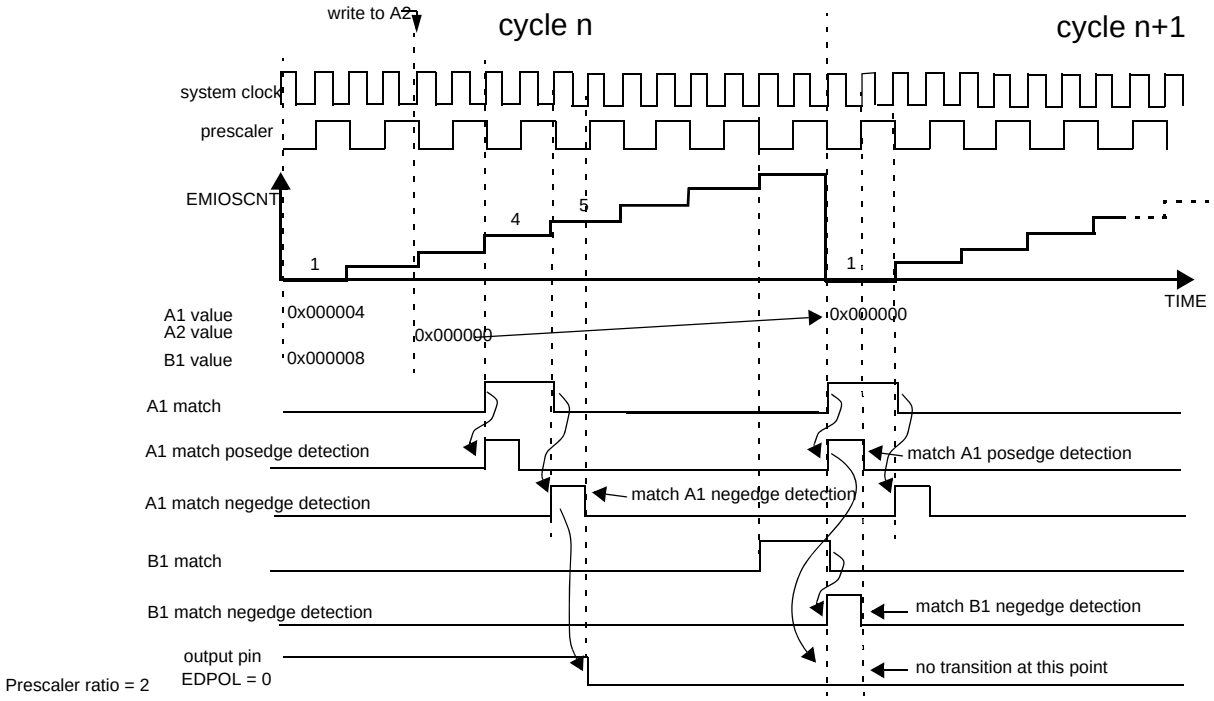


Figure 636. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 637 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 637* 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.

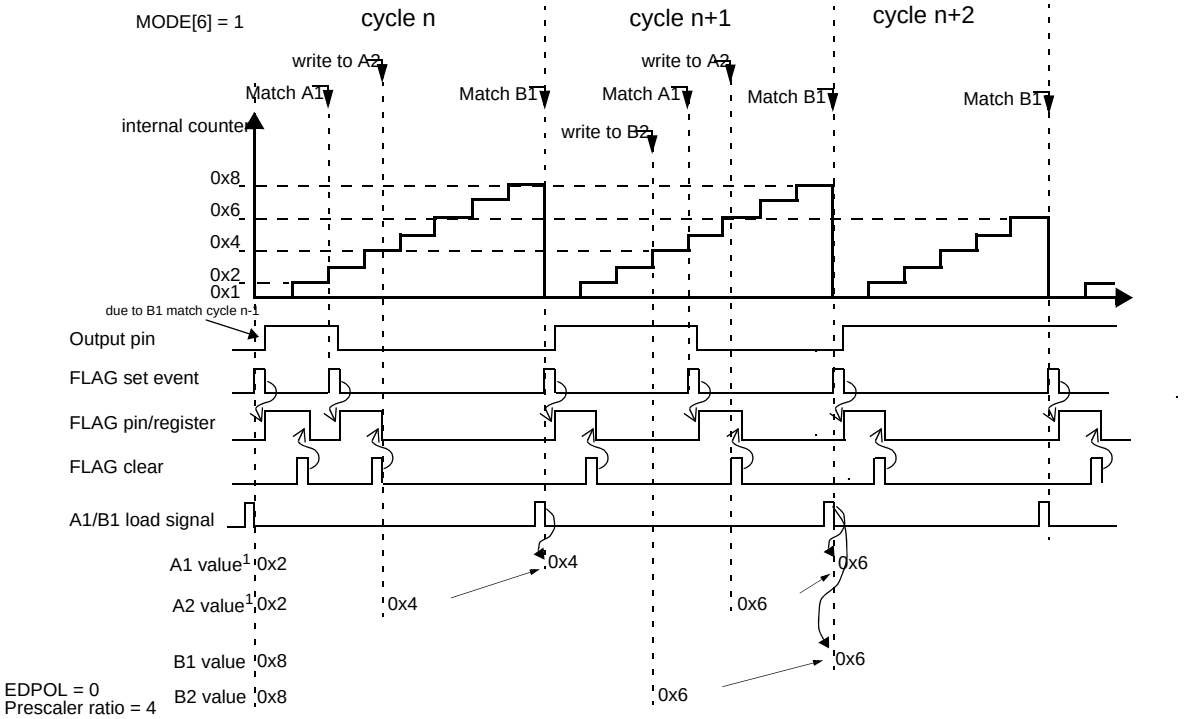


Figure 637. OPWFMB A1 and B1 registers update and flags

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 638 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.

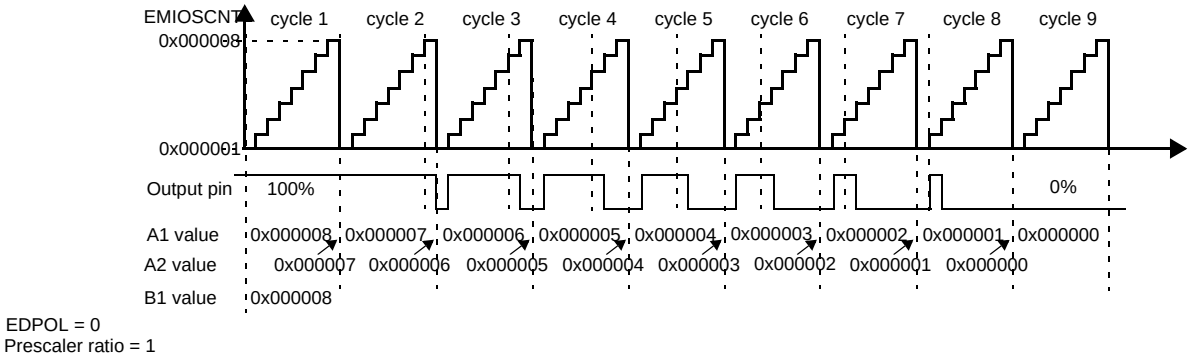


Figure 638. OPWFMB mode from 100% to 0% duty cycle

A 0% duty cycle signal is generated if A1 = 0x0 as shown in *Figure 638* 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 636* 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.

31.3.4.1.1.10 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 632*. 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 639 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**.

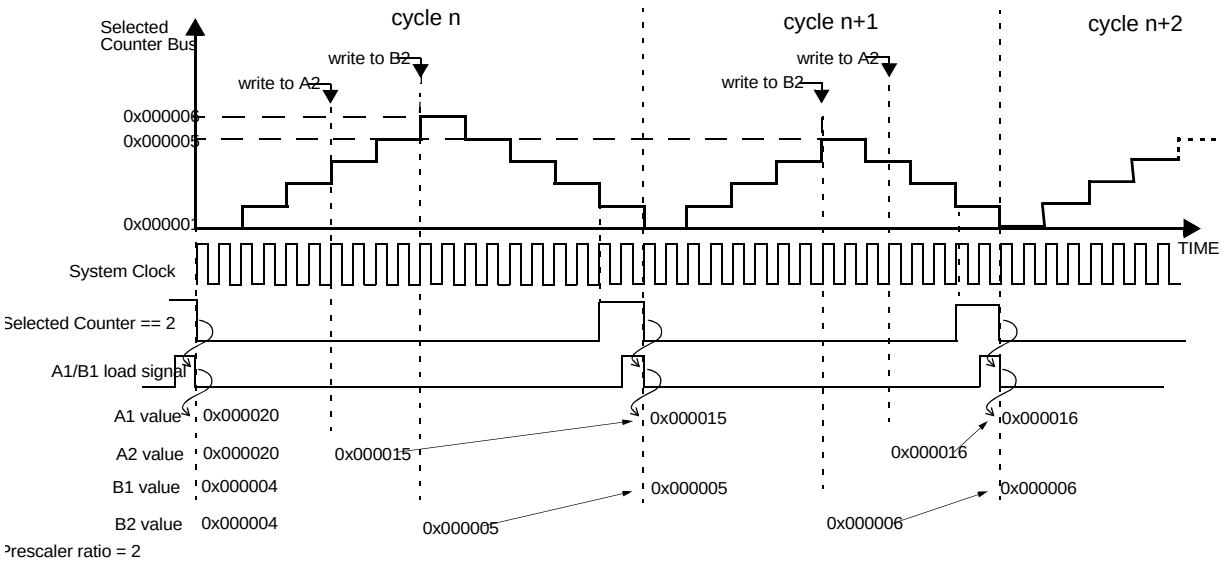


Figure 639. OPWMCB A1 and B1 registers load

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 640 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.

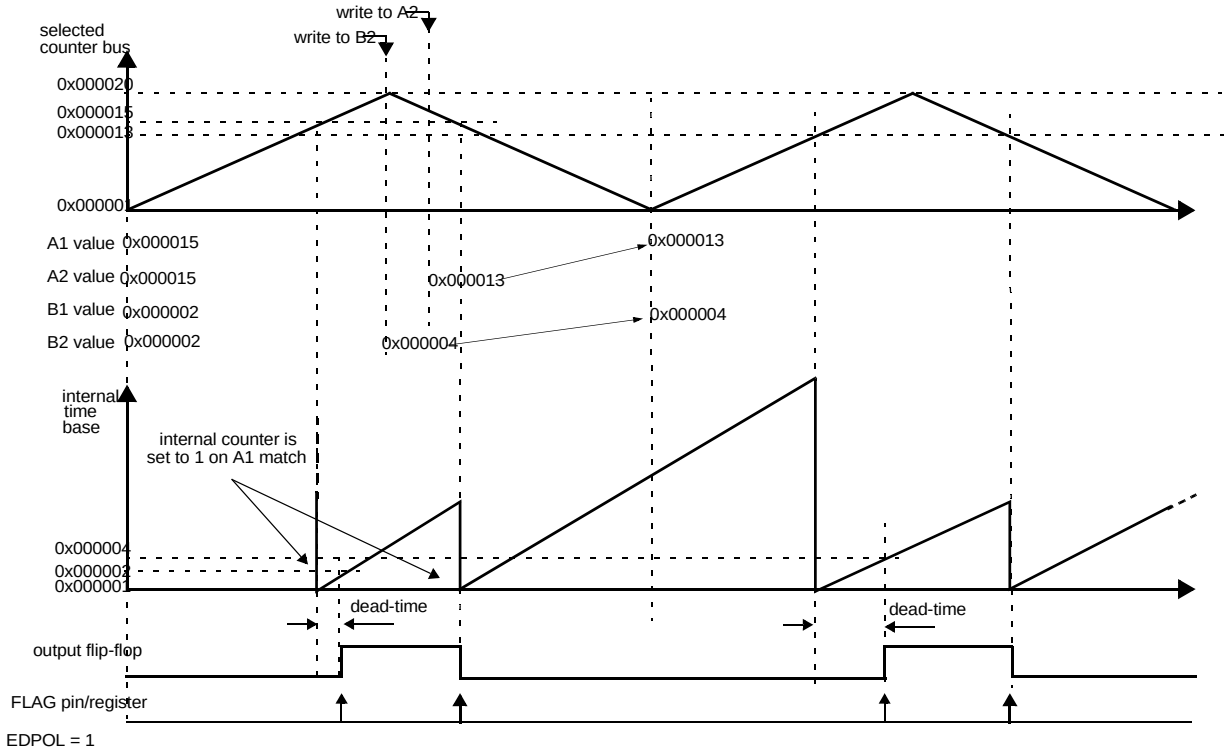


Figure 640. OPWMCB with lead dead time insertion

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.

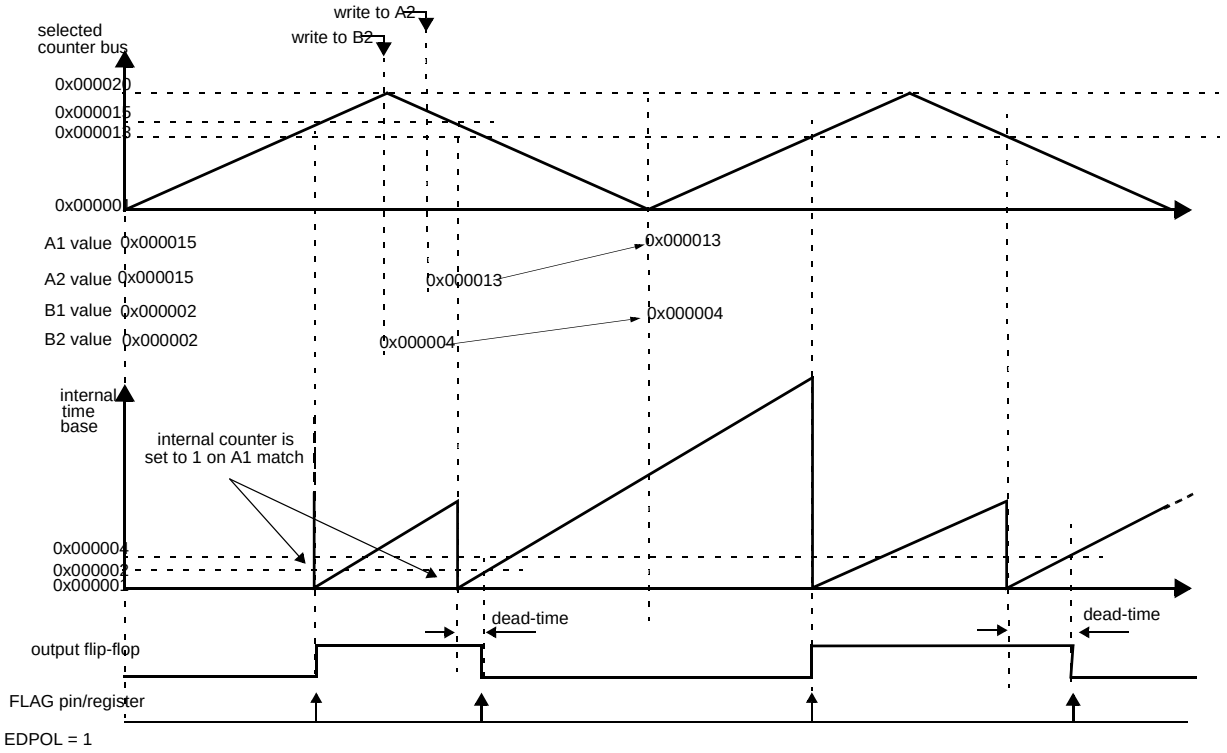


Figure 641. OPWMCB with trail dead time insertion

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 642 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.

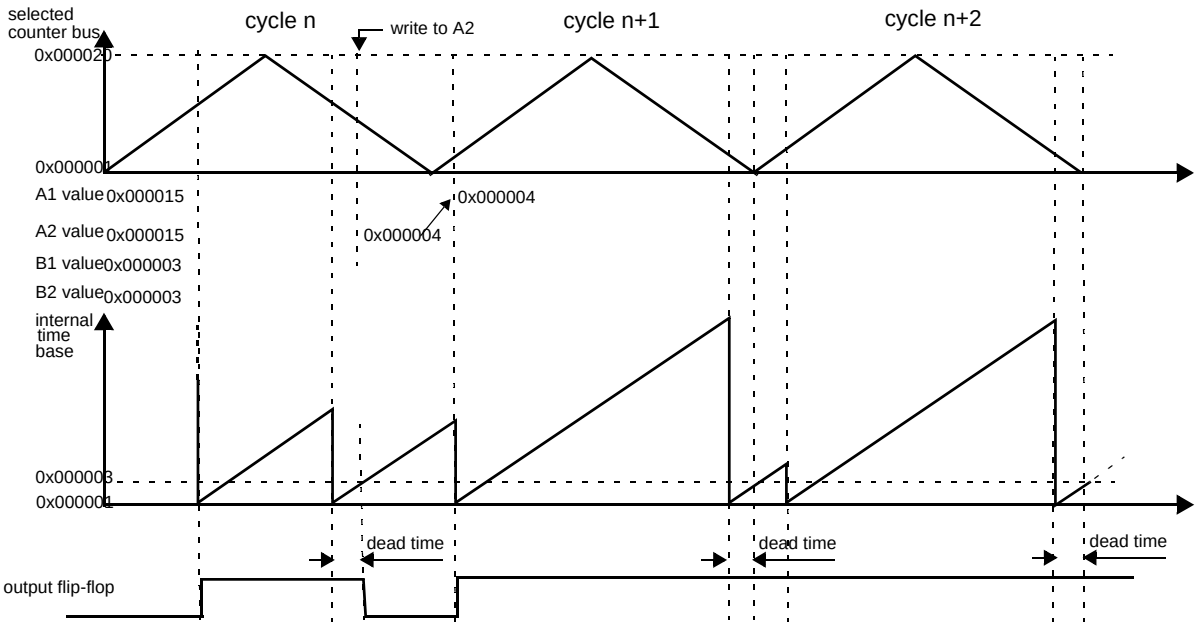


Figure 642. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3)

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 635* 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.

31.3.4.1.1.11 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 637* 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 643 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’.

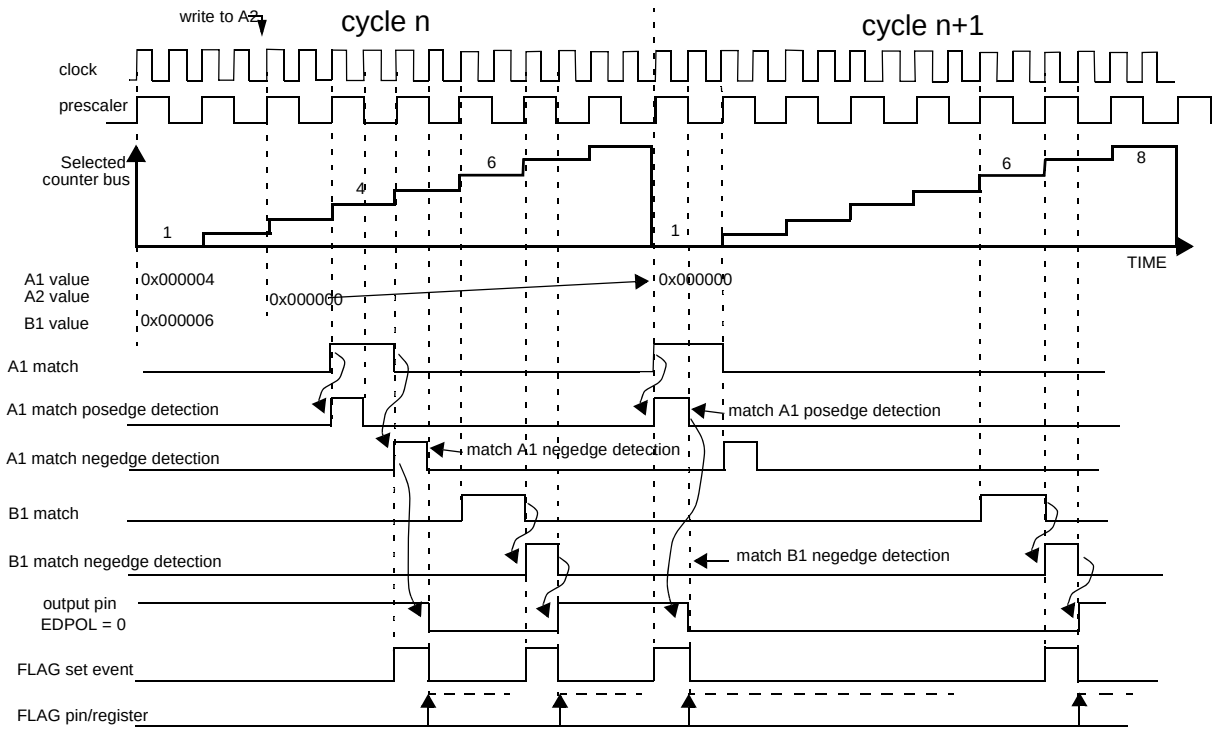


Figure 643. OPWMB mode matches and flags

Note that the output pin transitions are based on the negedges of the A1 and B1 match signals. *Figure 643* 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 644 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.

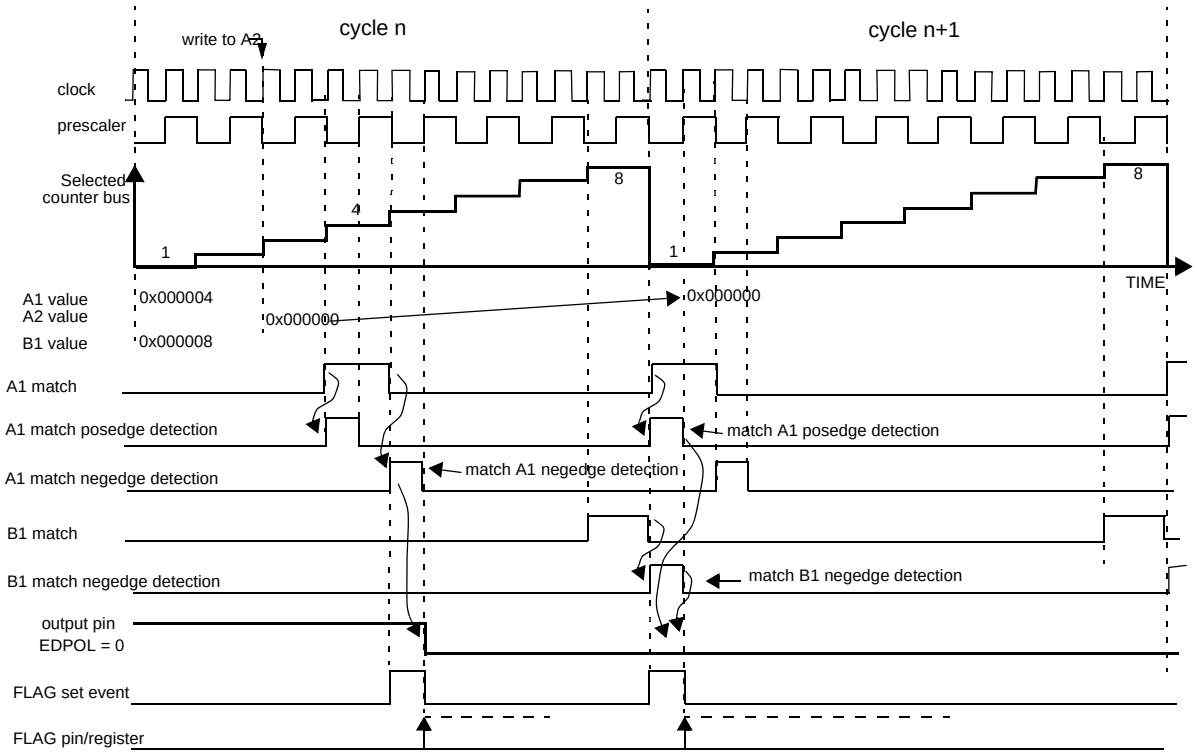


Figure 644. OPWMB mode with 0% duty cycle

Figure 645 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.

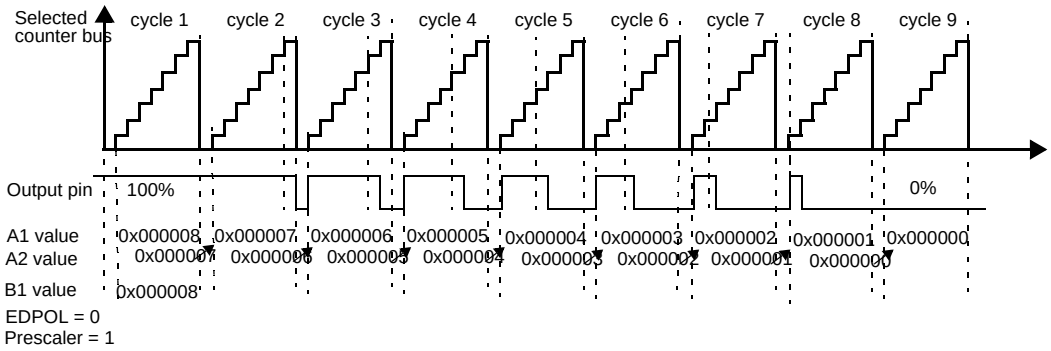


Figure 645. OPWMB mode from 100% to 0% duty cycle

In *Figure 645* 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.

31.3.4.1.1.12 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 643* at *Section 31.3.4.1.1.11: 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.

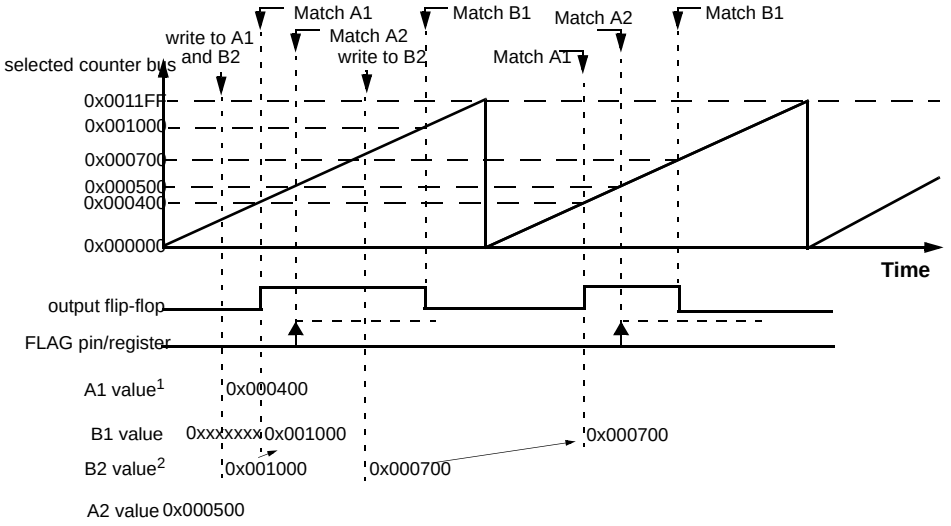
If subsequent matches occur on comparators A1 and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

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.

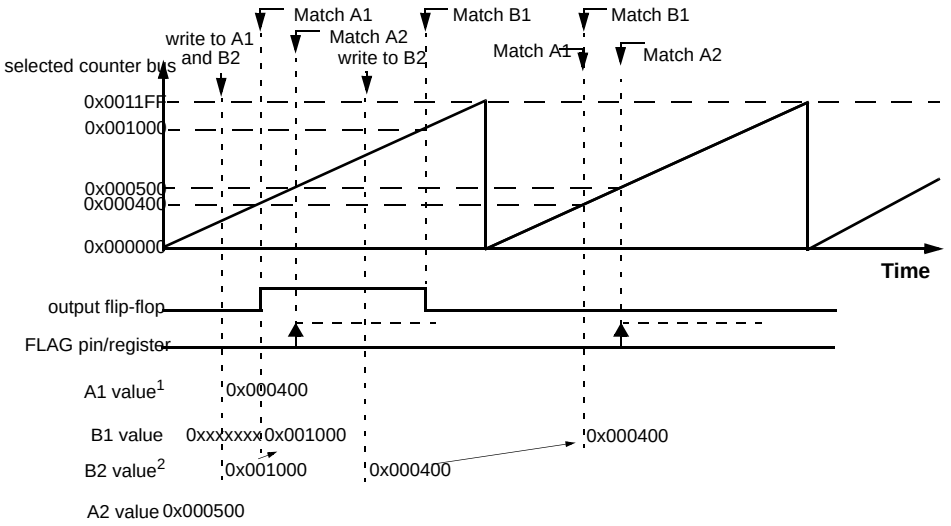
Figure 646 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and duty cycle update on next period update.



Notes1. EMIOSA[n] = A1
Notes2. EMIOSB[n] = B2 for write, B1 for read

Figure 646. OPWMT example

Figure 647 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 0% duty.



Notes1. EMIOSA[n] = A1
Notes2. EMIOSB[n] = B2 for write, B1 for read

Figure 647. OPWMT with 0% Duty Cycle

Figure 648 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 100% duty cycle.

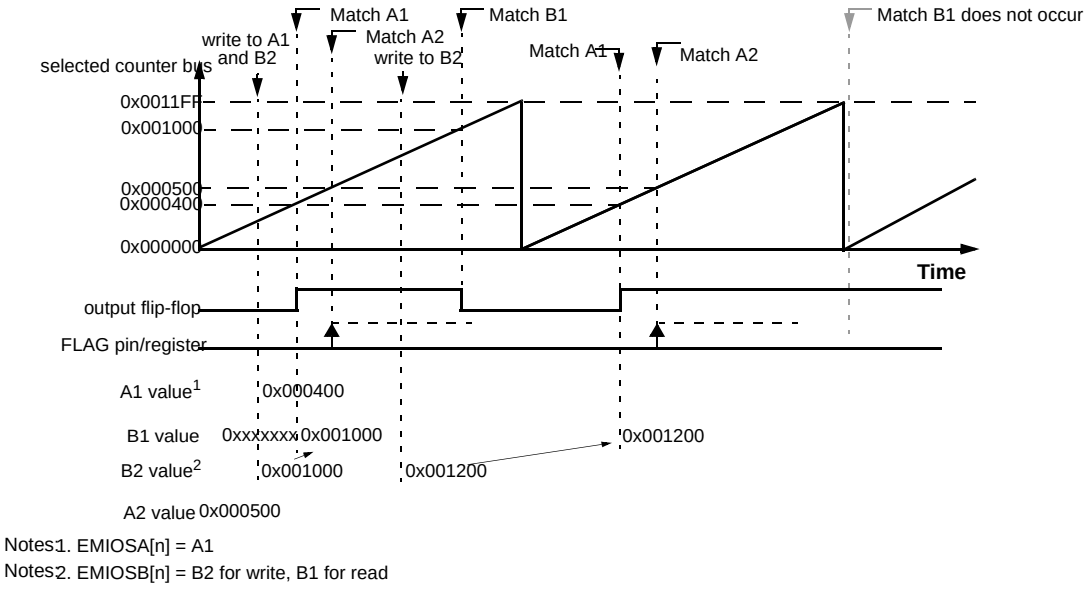


Figure 648. OPWMT with 100% duty cycle

31.3.4.1.2 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 649*.

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.

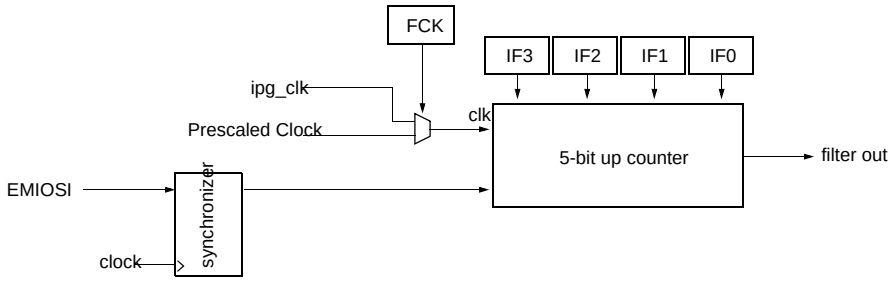


Figure 649. Input programmable filter submodule diagram

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 650*.

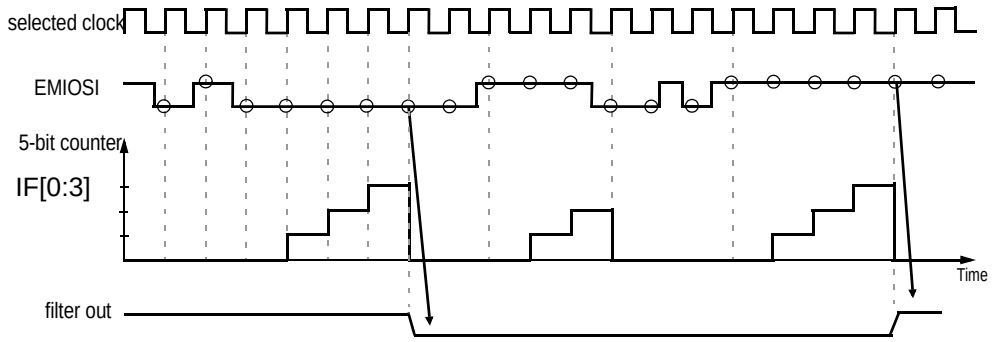


Figure 650. Input programmable filter example

The filter is not disabled during either freeze state or negated GTBE input.

31.3.4.1.3 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 550* 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.

31.3.4.1.4 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.

31.3.4.2 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.

31.3.4.2.1 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.

31.3.4.3 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 544* 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.

31.3.4.3.1 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.

31.3.5 Initialization/Application information

On resetting the eMIOS the Unified Channels enter GPIO input mode.

31.3.5.1 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.

31.3.5.2 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.

31.3.5.2.1 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 651* shows an example of a time base with prescaler ratio equal to one.

Note: MCB and OPWFMB modes have a different behavior.

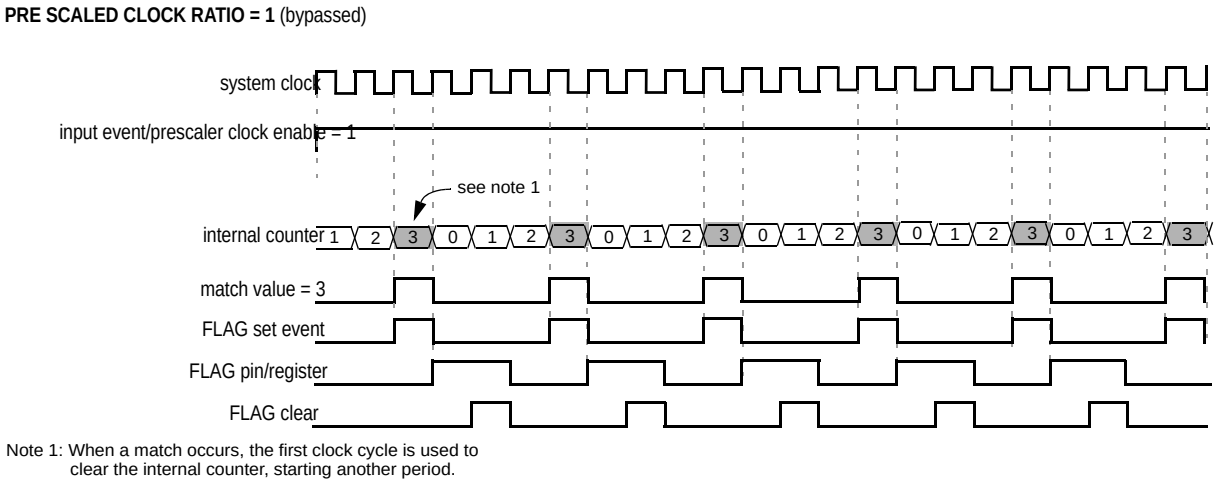
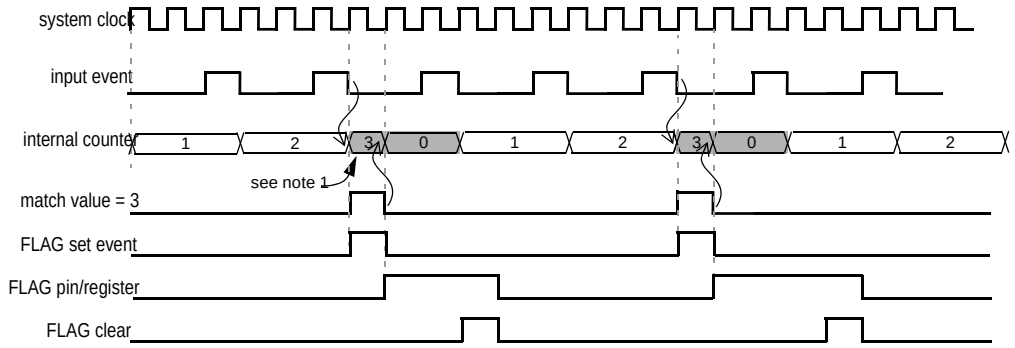


Figure 651. Time base period when running in the fastest prescaler ratio

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:

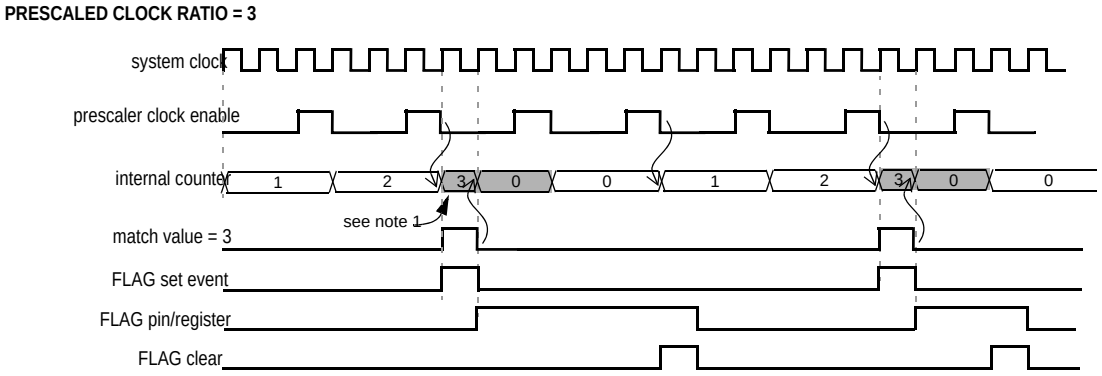
- If MC mode and Clear on Match Start and External Clock source are selected the internal counter behaves as described in *Figure 652*.
- If MC mode and Clear on Match Start and Internal Clock source are selected the internal counter behaves as described in *Figure 653*.
- If MC mode and Clear on Match End are selected the internal counter behaves as described in *Figure 654*.

Note: MCB and OPWFMB modes have a different behavior.



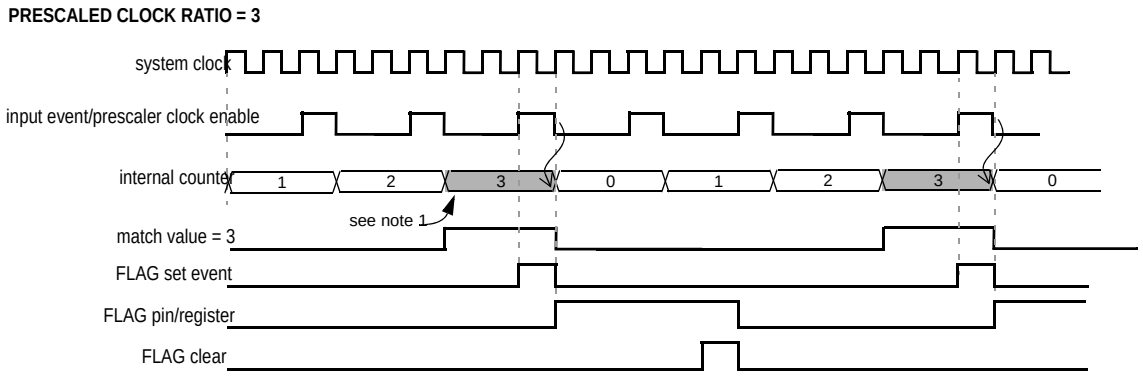
Note 1: When a match occurs, the first system clock cycle is used to clear the internal counter, and at the next edge of prescaler clock enable the counter will start counting.

Figure 652. 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 653. 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 654. Time base generation with clear on match end

31.3.5.2.2 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 requestor DMA 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.

31.3.5.2.3 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.

31.4 Periodic Interrupt Timer with Real-Time Interrupt (PIT_RTI)

31.4.1 Introduction

The PIT_RTI is an array of timers that can be used to raise interrupts and trigger DMA channels. The Real-Time Interrupt Timer (RTI) runs on a separate clock and can be used for system wakeup.

Figure 655 shows the PIT_RTI block diagram.

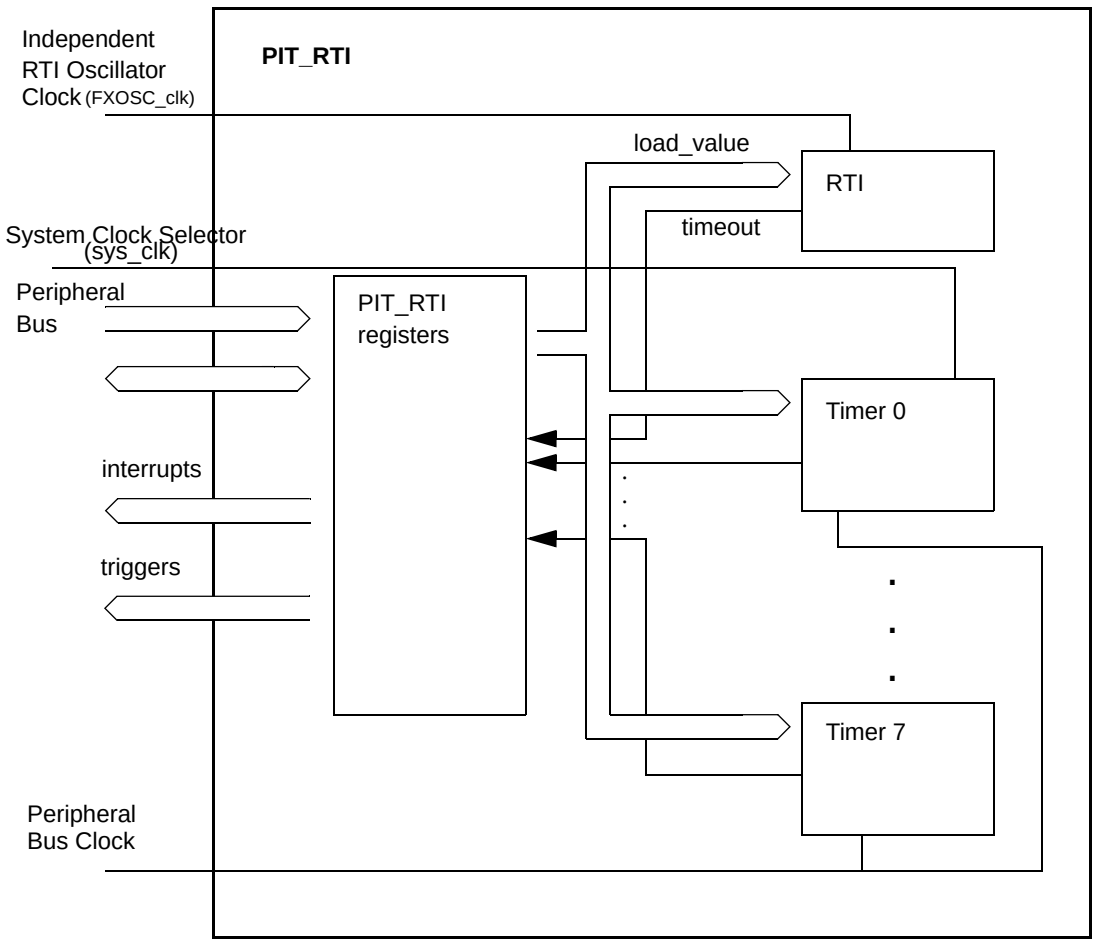


Figure 655. PIT_RTI block diagram

31.4.2 Features

The main features of this block are:

- One RTI (Real-Time Interrupt) timer to wakeup the CPU in stop mode
- Timers can generate DMA trigger pulses
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

31.4.3 Modes of operation

This subsection describes briefly all operating modes supported by the PIT_RTI.

- Run Mode
  All functional parts of the PIT_RTI are running during normal Run Mode.
- Stop Mode
  The PIT_RTI can continue to run in STOP mode.

31.4.4 Signal description

The PIT_RTI module has no external pins.

31.4.5 Memory map and register description

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

31.4.5.1 Memory map

Table 555 gives an overview of the PIT_RTI registers. See the chip memory map for the PIT_RTI base address.

Table 555. PIT_RTI memory map

Base address: 0xC3FF_0000		
Address offset	**Use**	**Location**
0x000	PIT_RTI Module Control Register (PITMCR)	*on page 1065*
0x004–0x0EC	Reserved	
0x0F0–0x0FC	RTI Channel	—
0x100–0x10C	Timer Channel 0	See *Table 556*
0x110–0x11C	Timer Channel 1	See *Table 556*
0x120–0x12C	Timer Channel 2	See *Table 556*
0x130–0x13C	Timer Channel 3	See *Table 556*
0x140–0x14C	Timer Channel 4	See *Table 556*
0x150–0x15C	Timer Channel 5	See *Table 556*

Table 555. PIT_RTI memory map (continued)

Base address: 0xC3FF_0000		
Address offset	**Use**	**Location**
0x160–0x16C	Timer Channel 6	See *Table 556*
0x170–0x17C	Timer Channel 7	See *Table 556*

Table 556. Timer channel *n* / RTI channel

Address offset	Use	Location
channel + 0x00	Timer Load Value Register (LDVAL)	*on page 1066*
channel + 0x04	Current Timer Value Register (CVAL)	*on page 1066*
channel + 0x08	Timer Control Register (TCTRL)	*on page 1067*
channel + 0x0C	Timer Flag Register (TFLG)	*on page 1068*

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.

Note: The RTI registers should be programmed only when the PIT_RTI clock is running.

31.4.5.2 PIT_RTI Module Control Register (PITMCR)

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

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	MDIS_RTI	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 656. PIT_RTI Module Control Register (PITMCR)

Table 557. PITMCR field descriptions

Field	Description
MDIS_RTI	Module Disable - RTI section. This is used to disable the RTI timer. This bit should be enabled before any RTI setup is done. 0 Clock for RTI is enabled 1 Clock for RTI disabled(default)
MDIS	Module Disable - (PIT section). This is used to disable the standard timers. The RTI timer is not affected by this bit. 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.

31.4.5.3 Timer Load Value Register (LDVAL)

This register selects the timeout period for the timer interrupts. In the case of the RTI, it will take several cycles until this value is synchronized into the RTI clock domain. For all other timers the value change is visible immediately. The synchronization mechanism allows 0 wait states in this case.

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

Figure 657. Timer Load Value Register (LDVAL)

Table 558. 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 662*).

31.4.5.4 Current Timer Value Register (CVAL)

This register indicates the current timer position. In the case of the RTI, this will show a value which is several cycles old, since it originates from a potentially different clock domain.

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

Figure 658. Current Timer Value Register (CVAL)

Table 559. 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_RTI Module Control Register (see *Figure 601*).

31.4.5.5 Timer Control Register (TCTRL)

This register contains the control bits for each timer.

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

Figure 659. Timer Control Register (TCTRL)

Table 560. 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

31.4.5.6 Timer Flag Register (TFLG)

This register holds the PIT_RTI interrupt flags.

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

Figure 660. Timer Flag Register (TFLG)

Table 561. 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

31.4.6 Functional description

31.4.6.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line.

31.4.6.1.1 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 661*).

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 662*).

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 663*).

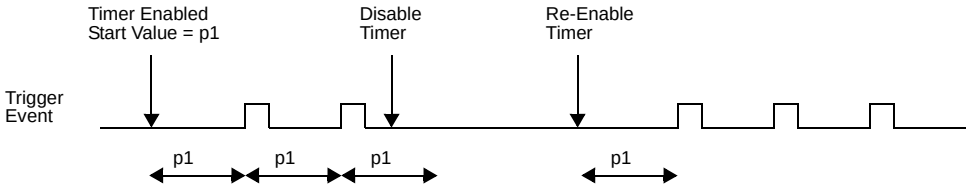


Figure 661. Stopping and starting a timer

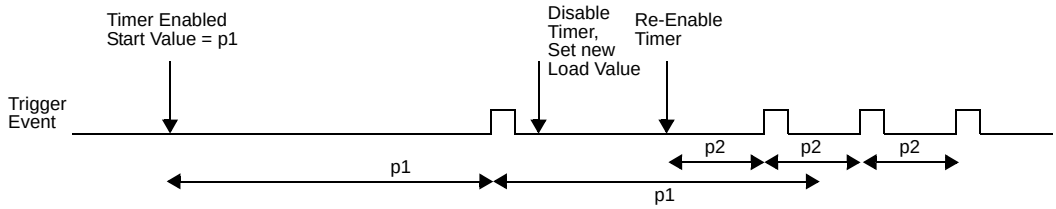


Figure 662. Modifying running timer period

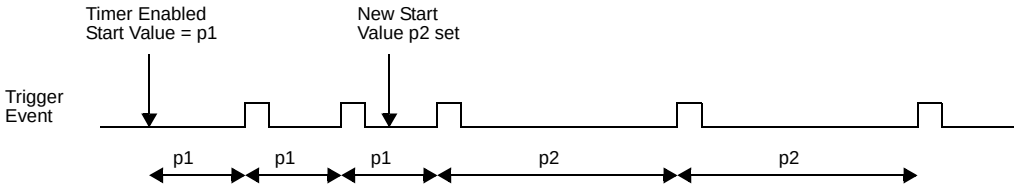


Figure 663. Dynamically setting a new load value

31.4.6.1.2 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.

31.4.6.2 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.

31.4.7 Initialization and application information

31.4.7.1 Example configuration

In the example configuration:

- The PIT_RTI 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_RTI 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_RTI
PIT_RTI_CTRL = 0x00;
// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL = PIT_TIE; // let RTI generate interrupts
PIT_RTI_TCTRL |= PIT_TEN; // start RTI

// Timer 1
PIT_RTI_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_RTI_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_RTI_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_RTI_LDVAL3 = 0x0016E35F; // setup timer 3 for 1500000 cycles
PIT_RTI_TCTRL3 = TEN; // start timer 3
```

32 Analog-to-Digital Converter (ADC)

32.1 Overview

32.1.1 Device-specific pin configuration features

The device comprises two ADC modules:

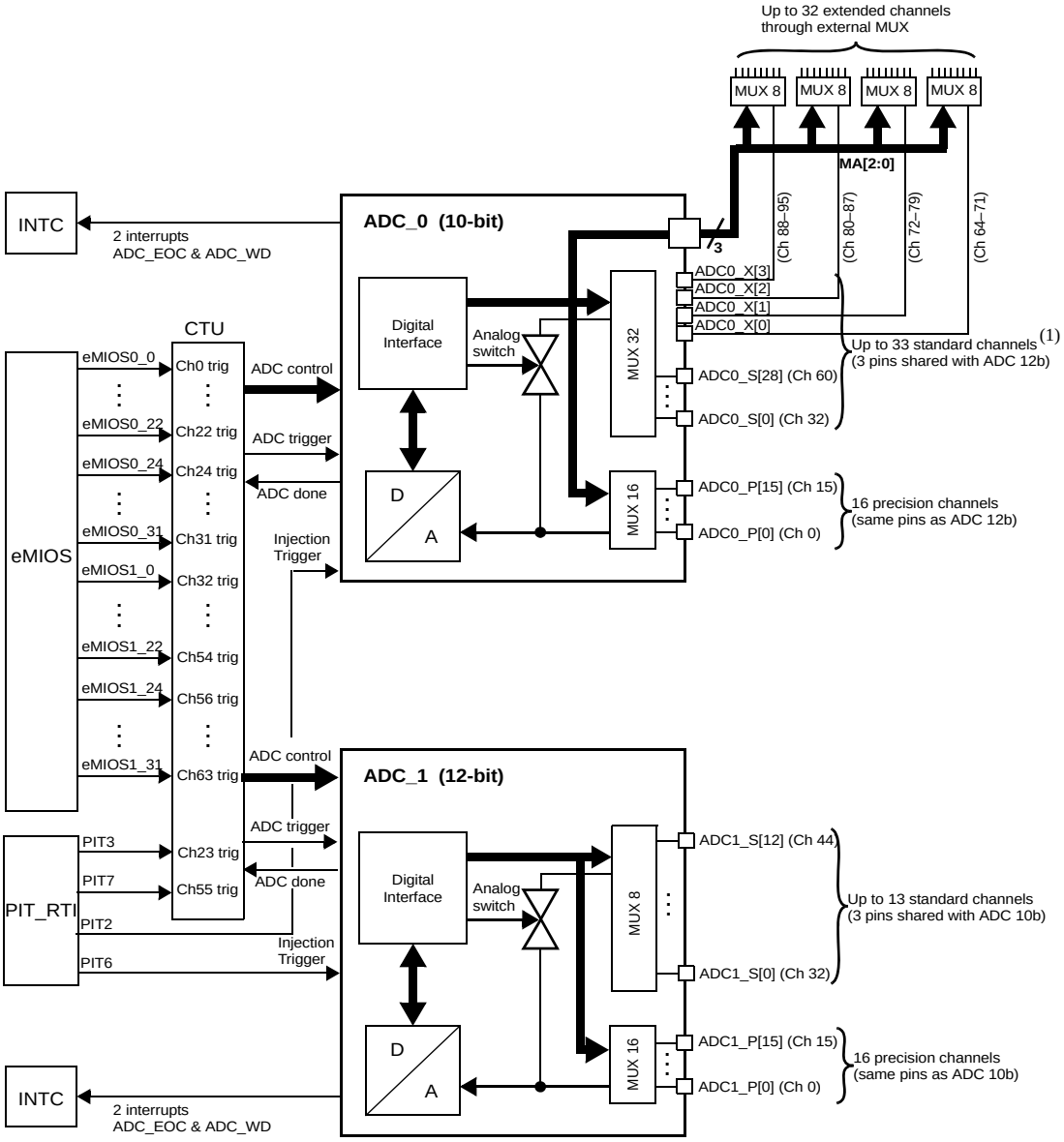
- ADC_0 with 10-bit resolution
- ADC_1 with 12-bit resolution
- Common mode conversion range
  - For ADC_0: 0 to VDD_HV_ADC0
  - For ADC_1: 0 to VDD_HV_ADC1
- Independent reference supplies for each ADC
- 62 single-ended input channels (depending on package type), expandable to 90 channels via external multiplexing
  - Internally multiplexed channels
    - 16 precision channels shared between 10-bit and 12-bit ADCs
    - 3 standard channels[t] shared between 10-bit and 12-bit ADCs
    - 10 dedicated standard channels on 12-bit ADC
    - Up to 29[u] dedicated standard channels on 10-bit ADC
  - Externally multiplexed channels are dedicated to 10-bit ADC
    - Internal control to support generation of external analog multiplexer selection
    - 4 internal channels optionally used to support externally multiplexed inputs, providing transparent control for additional ADC channels
    - Each of the 4 channels supports as many as 8 externally multiplexed inputs
- 3 independently configurable sample and conversion times for high precision channels, standard precision channels and externally multiplexed channels
- Dedicated result registers available for every channel. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- One Shot/Scan Modes
- Chain Injection Mode
- Conversion triggering sources:
  - Software
  - CTU
  - PIT_RTI channel 2 and 6 (for injected conversion)
- Conversion triggering support — Internal conversion triggering from PIT_RTI or timed I/O module (eMIOS)

t. Standard channels are mapped on PB8, PB9, and PB10.

u. There are 29 standard channels dedicated to ADC 10-bit, and there are 4 dedicated channels on ADC 10 bit that "support external mux" option. If they are NOT used with external mux, they can be used as standard channel and hence the total number of standard channels dedicated to 10 bit ADC is 33. If they are muxed, then there are 29 standard channels and 4 extended channels.

- Power-down mode for analog portion of ADC
- Supports DMA transfer of results based on the end of conversion
- 6 + 3 analog watchdogs (6 on 10-bit ADC, 3 on 12-bit ADC) with interrupt capability for continuous hardware monitoring

32.1.2 Device-specific implementation



1 There are 29 standard channels dedicated to ADC 10-bit, and thereare 4 dedicated channels on ADC 10 bit that "support external mux" option. If they are NOT used with external mux, they can be used as standard channel and hence the total number of standard channels dedicated to 10 bit ADC is 33 . If they are muxed, then there are 29 standard channels and 4 extended channels.

Figure 664. Implementation of ADC_0 and ADC_1

Table 562. ADC channel mapping

Pad	ADC_0 (10-bit)		ADC_1 (12-bit)		Description
	Function	Channel #	Function	Channel #	
PB[4]	ADC0_P[0]	CH0	ADC1_P[0]	CH0	16 precision channels shared between ADC_0 and ADC_1
PB[5]	ADC0_P[1]	CH1	ADC1_P[1]	CH1	
PB[6]	ADC0_P[2]	CH2	ADC1_P[2]	CH2	
PB[7]	ADC0_P[3]	CH3	ADC1_P[3]	CH3	
PD[0]	ADC0_P[4]	CH4	ADC1_P[4]	CH4	
PD[1]	ADC0_P[5]	CH5	ADC1_P[5]	CH5	
PD[2]	ADC0_P[6]	CH6	ADC1_P[6]	CH6	
PD[3]	ADC0_P[7]	CH7	ADC1_P[7]	CH7	
PD[4]	ADC0_P[8]	CH8	ADC1_P[8]	CH8	
PD[5]	ADC0_P[9]	CH9	ADC1_P[9]	CH9	
PD[6]	ADC0_P[10]	CH10	ADC1_P[10]	CH10	
PD[7]	ADC0_P[11]	CH11	ADC1_P[11]	CH11	
PD[8]	ADC0_P[12]	CH12	ADC1_P[12]	CH12	
PD[9]	ADC0_P[13]	CH13	ADC1_P[13]	CH13	
PD[10]	ADC0_P[14]	CH14	ADC1_P[14]	CH14	
PD[11]	ADC0_P[15]	CH15	ADC1_P[15]	CH15	
PB[8]	ADC0_S[0]	CH32	ADC1_S[4]	CH36	3 standard channels shared between ADC_0 and ADC_1
PB[9]	ADC0_S[1]	CH33	ADC1_S[5]	CH37	
PB[10]	ADC0_S[2]	CH34	ADC1_S[6]	CH38	

Table 562. ADC channel mapping (continued)

Pad	ADC_0 (10-bit)		ADC_1 (12-bit)		Description
	Function	Channel #	Function	Channel #	
PB[11]	ADC0_S[3]	CH35			29 standard channels dedicated to ADC_0
PD[12]	ADC0_S[4]	CH36			
PD[13]	ADC0_S[5]	CH37			
PD[14]	ADC0_S[6]	CH38			
PD[15]	ADC0_S[7]	CH39			
PF[0]	ADC0_S[8]	CH40			
PF[1]	ADC0_S[9]	CH41			
PF[2]	ADC0_S[10]	CH42			
PF[3]	ADC0_S[11]	CH43			
PF[4]	ADC0_S[12]	CH44			
PF[5]	ADC0_S[13]	CH45			
PF[6]	ADC0_S[14]	CH46			
PF[7]	ADC0_S[15]	CH47			
PI[8]	ADC0_S[16]	CH48			
PI[9](1)	ADC0_S[17]	CH49			
PI[10]1	ADC0_S[18]	CH50			
PI[11]	ADC0_S[19]	CH51			
PI[12]	ADC0_S[20]	CH52			
PI[13]	ADC0_S[21]	CH53			
PI[14]	ADC0_S[22]	CH54			
PI[15]	ADC0_S[23]	CH55			
PJ[0]	ADC0_S[24]	CH56			
PJ[1]	ADC0_S[25]	CH57			
PJ[2]	ADC0_S[26]	CH58			
PJ[3]	ADC0_S[27]	CH59			
PJ[5]1	ADC0_S[28]	CH60			
PJ[6]1	ADC0_S[29]	CH61			
PJ[7]1	ADC0_S[30]	CH62			
PJ[8]1	ADC0_S[31]	CH63			

Table 562. ADC channel mapping (continued)

Pad	ADC_0 (10-bit)		ADC_1 (12-bit)		Description
	Function	Channel #	Function	Channel #	
PA[3]			ADC1_S[0]	CH32	10 standard channels dedicated to ADC_1
PA[7]			ADC1_S[1]	CH33	
PA[10]			ADC1_S[2]	CH34	
PA[11]			ADC1_S[3]	CH35	
PE[12]			ADC1_S[7]	CH39	
PJ[9][1]			ADC1_S[8]	CH40	
PJ[10][1]			ADC1_S[9]	CH41	
PJ[11][1]			ADC1_S[10]	CH42	
PJ[12][1]			ADC1_S[11]	CH43	
PJ[13][1]			ADC1_S[12]	CH44	
PB[12]	ADC0_X[0]	CH[64..71]			4 muxed channels dedicated to ADC_0
PB[13]	ADC0_X[1]	CH[72..79]			
PB[14]	ADC0_X[2]	CH[80..87]			
PB[15]	ADC0_X[3]	CH[88..95]			

1. This pin is not available in the 176 LQFP package

Table 563. ADC_0 mux control signal availability

Control signal	Pads
MA[0]	PC[3], PE[7], PH[8]
MA[1]	PC[10], PE[6], PH[7]
MA[2]	PA[2], PE[5], PH[6]

32.2 Introduction

The analog-to-digital converter (ADC) block provides accurate and fast conversions for a wide range of applications. An ADC module has a corresponding digital interface.

The ADC digital interface contains advanced features for normal or injected conversion. It provides support for eDMA (direct memory access) mode operation. A conversion can be triggered by software or hardware (Cross Triggering Unit or PIT_RTI).

There three types of input channels:

- Internal precision, ADCx_P[n] (internally multiplexed high accuracy precision channels)
- Internal standard, ADCx_S[n] (internally multiplexed standard accuracy channels)
- External ADCx_X[n] (externally multiplexed standard accuracy channels)

The mask registers present within the ADCDig 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.

Note: Configure all the GPIOs in a known state during 12-bit ADC conversion for better performance.

32.3 Register descriptions

32.3.1 Introduction

SPC564Bxx and SPC56ECxx has two ADCs (10-bit ADC_0 and 12-bit ADC_1) and each has specific registers.

Table 564 lists the ADC_0 registers with their address offsets and reset values.

Table 564. 10-bit ADC_0 digital registers

Base address: 0xFFE0_0000		
Register name	**Address offset**	**Location**
Main Configuration Register (MCR)	0x0000	*on page 1083*
Main Status Register (MSR)	0x0004	*on page 1086*
Reserved	0x0008 .. 0x000F	
Interrupt Status Register (ISR)	0x0010	*on page 1086*
Channel Pending Register (CEOCFR0)	0x0014	*on page 1087*
Channel Pending Register (CEOCFR1)	0x0018	*on page 1087*
Channel Pending Register (CEOCFR2)	0x001C	*on page 1087*
Interrupt Mask Register (IMR)	0x0020	*on page 1090*
Channel Interrupt Mask Register (CIMR0)	0x0024	*on page 1090*
Channel Interrupt Mask Register (CIMR1)	0x0028	*on page 1090*
Channel Interrupt Mask Register (CIMR2)	0x002C	*on page 1090*
Watchdog Threshold Interrupt Status Register (WTISR)	0x0030	*on page 1092*
Watchdog Threshold Interrupt Mask Register (WTIMR)	0x0034	*on page 1093*
Reserved	0x0038 .. 0x003F	
DMA Enable Register (DMAE)	0x0040	*on page 1094*
DMA Channel Select Register 0 (DMAR0)	0x0044	*on page 1095*
DMA Channel Select Register 1 (DMAR1)	0x0048	*on page 1095*
DMA Channel Select Register 2 (DMAR2)	0x004C	*on page 1095*
Reserved	0x0050 .. 0x005F	
Threshold Register 0 (THRHLR0)	0x0060	*on page 1097*

Table 564. 10-bit ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		
Register name	**Address offset**	**Location**
Threshold Register 1 (THRHLR1)	0x0064	*on page 1097*
Threshold Register 2 (THRHLR2)	0x0068	*on page 1097*
Threshold Register 3 (THRHLR3)	0x006C	*on page 1097*
Reserved	0x0070 .. 0x007F	
Presampling Control Register (PSCR)	0x0080	*on page 1099*
Presampling Register 0 (PSR0)	0x0084	*on page 1099*
Presampling Register 1 (PSR1)	0x0088	*on page 1099*
Presampling Register 2 (PSR2)	0x008C	*on page 1099*
Reserved	0x0090 .. 0x0093	
Conversion Timing Register 0 (CTR0)	0x0094	*on page 1101*
Conversion Timing Register 1 (CTR1)	0x0098	*on page 1101*
Conversion Timing Register 2 (CTR2)	0x009C	*on page 1101*
Reserved	0x00A0 .. 0x00A3	
Normal Conversion Mask Register 0 (NCMR0)	0x00A4	*on page 1102*
Normal Conversion Mask Register 1 (NCMR1)	0x00A8	*on page 1102*
Normal Conversion Mask Register 2 (NCMR2)	0x00AC	*on page 1102*
Reserved	0x00B0 .. 0x00B3	
Injected Conversion Mask Register 0 (JCMR0)	0x00B4	*on page 1104*
Injected Conversion Mask Register 1 (JCMR1)	0x00B8	*on page 1104*
Injected Conversion Mask Register 2 (JCMR2)	0x00BC	*on page 1104*
Reserved	0x00C0 .. 0x00C3	
Decode Signal Delay Register (DSDR)	0x00C4	*on page 1106*
Power-down Exit Delay Register (PDEDR)	0x00C8	*on page 1107*
Reserved	0x00CC .. 0x00FF	
Channel 0 Data Register (CDR0)	0x0100	*on page 1108*
Channel 1 Data Register (CDR1)	0x0104	*on page 1108*
Channel 2 Data Register (CDR2)	0x0108	*on page 1108*
Channel 3 Data Register (CDR3)	0x010C	*on page 1108*
Channel 4 Data Register (CDR4)	0x0110	*on page 1108*
Channel 5 Data Register (CDR5)	0x0114	*on page 1108*
Channel 6 Data Register (CDR6)	0x0118	*on page 1108*
Channel 7 Data Register (CDR7)	0x011C	*on page 1108*
Channel 8 Data Register (CDR8)	0x0120	*on page 1108*

Table 564. 10-bit ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		
Register name	**Address offset**	**Location**
Channel 9 Data Register (CDR9)	0x0124	*on page 1108*
Channel 10 Data Register (CDR10)	0x0128	*on page 1108*
Channel 11 Data Register (CDR11)	0x012C	*on page 1108*
Channel 12 Data Register (CDR12)	0x0130	*on page 1108*
Channel 13 Data Register (CDR13)	0x0134	*on page 1108*
Channel 14 Data Register (CDR14)	0x0138	*on page 1108*
Channel 15 Data Register (CDR15)	0x013C	*on page 1108*
Reserved	0x0140 .. 0x017F	
Channel 32 Data Register (CDR32)	0x0180	*on page 1108*
Channel 33 Data Register (CDR33)	0x0184	*on page 1108*
Channel 34 Data Register (CDR34)	0x0188	*on page 1108*
Channel 35 Data Register (CDR35)	0x018C	*on page 1108*
Channel 36 Data Register (CDR36)	0x0190	*on page 1108*
Channel 37 Data Register (CDR37)	0x0194	*on page 1108*
Channel 38 Data Register (CDR38)	0x0198	*on page 1108*
Channel 39 Data Register (CDR39)	0x019C	*on page 1108*
Channel 40 Data Register (CDR40)	0x01A0	*on page 1108*
Channel 41 Data Register (CDR41)	0x01A4	*on page 1108*
Channel 42 Data Register (CDR42)	0x01A8	*on page 1108*
Channel 43 Data Register (CDR43)	0x01AC	*on page 1108*
Channel 44 Data Register (CDR44)	0x01B0	*on page 1108*
Channel 45 Data Register (CDR45)	0x01B4	*on page 1108*
Channel 46 Data Register (CDR46)	0x01B8	*on page 1108*
Channel 47 Data Register (CDR47)	0x01BC	*on page 1108*
Channel 48 Data Register (CDR48)	0x01C0	*on page 1108*
Channel 49 Data Register (CDR49)	0x01C4	*on page 1108*
Channel 50 Data Register (CDR50)	0x01C8	*on page 1108*
Channel 51 Data Register (CDR51)	0x01CC	*on page 1108*
Channel 52 Data Register (CDR52)	0x01D0	*on page 1108*
Channel 53 Data Register (CDR53)	0x01D4	*on page 1108*
Channel 54 Data Register (CDR54)	0x01D8	*on page 1108*
Channel 55 Data Register (CDR55)	0x01DC	*on page 1108*
Channel 56 Data Register (CDR56)	0x01E0	*on page 1108*

Table 564. 10-bit ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		
Register name	**Address offset**	**Location**
Channel 57 Data Register (CDR57)	0x01E4	*on page 1108*
Channel 58 Data Register (CDR58)	0x01E8	*on page 1108*
Channel 59 Data Register (CDR59)	0x01EC	*on page 1108*
Channel 60 Data Register (CDR60)	0x01F0	*on page 1108*
Channel 61 Data Register (CDR61)	0x01F4	*on page 1108*
Channel 62 Data Register (CDR62)	0x01F8	*on page 1108*
Channel 63 Data Register (CDR63)	0x01FC	*on page 1108*
Channel 64 Data Register (CDR64)	0x0200	*on page 1108*
Channel 65 Data Register (CDR65)	0x0204	*on page 1108*
Channel 66 Data Register (CDR66)	0x0208	*on page 1108*
Channel 67 Data Register (CDR67)	0x020C	*on page 1108*
Channel 68 Data Register (CDR68)	0x0210	*on page 1108*
Channel 69 Data Register (CDR69)	0x0214	*on page 1108*
Channel 70 Data Register (CDR70)	0x0218	*on page 1108*
Channel 71 Data Register (CDR71)	0x021C	*on page 1108*
Channel 72 Data Register (CDR72)	0x0220	*on page 1108*
Channel 73 Data Register (CDR73)	0x0224	*on page 1108*
Channel 74 Data Register (CDR74)	0x0228	*on page 1108*
Channel 75 Data Register (CDR75)	0x022C	*on page 1108*
Channel 76 Data Register (CDR76)	0x0230	*on page 1108*
Channel 77 Data Register (CDR77)	0x0234	*on page 1108*
Channel 78 Data Register (CDR78)	0x0238	*on page 1108*
Channel 79 Data Register (CDR79)	0x023C	*on page 1108*
Channel 80 Data Register (CDR80)	0x0240	*on page 1108*
Channel 81 Data Register (CDR81)	0x0244	*on page 1108*
Channel 82 Data Register (CDR82)	0x0248	*on page 1108*
Channel 83 Data Register (CDR83)	0x024C	*on page 1108*
Channel 84 Data Register (CDR84)	0x0250	*on page 1108*
Channel 85 Data Register (CDR85)	0x0254	*on page 1108*
Channel 86 Data Register (CDR86)	0x0258	*on page 1108*
Channel 87 Data Register (CDR87)	0x025C	*on page 1108*
Channel 88 Data Register (CDR88)	0x0260	*on page 1108*
Channel 89 Data Register (CDR89)	0x0264	*on page 1108*

Table 564. 10-bit ADC_0 digital registers (continued)

Base address: 0xFFE0_0000		
Register name	**Address offset**	**Location**
Channel 90 Data Register (CDR90)	0x0268	*on page 1108*
Channel 91 Data Register (CDR91)	0x026C	*on page 1108*
Channel 92 Data Register (CDR92)	0x0270	*on page 1108*
Channel 93 Data Register (CDR93)	0x0274	*on page 1108*
Channel 94 Data Register (CDR94)	0x0278	*on page 1108*
Channel 95 Data Register (CDR95)	0x027C	*on page 1108*
Threshold Register 4 (THRHLR4)	0x0280	*on page 1097*
Threshold Register 5 (THRHLR5)	0x0284	*on page 1097*
Reserved	0x0288 .. 0x02AF	
Channel Watchdog Selection Register 0 (CWSELR0)	0x02B0	*on page 1109*
Channel Watchdog Selection Register 1 (CWSELR1)	0x02B4	*on page 1109*
Reserved	0x02B8 .. 0x02BF	
Channel Watchdog Selection Register 4 (CWSELR4)	0x02C0	*on page 1109*
Channel Watchdog Selection Register 5 (CWSELR5)	0x02C4	*on page 1109*
Channel Watchdog Selection Register 6 (CWSELR6)	0x02C8	*on page 1109*
Channel Watchdog Selection Register 7 (CWSELR7)	0x02CC	*on page 1109*
Channel Watchdog Selection Register 8 (CWSELR8)	0x02D0	*on page 1109*
Channel Watchdog Selection Register 9 (CWSELR9)	0x02D4	*on page 1109*
Channel Watchdog Selection Register 10 (CWSELR10)	0x02D8	*on page 1109*
Channel Watchdog Selection Register 11 (CWSELR11)	0x02DC	*on page 1109*
Channel Watchdog Enable Register 0 (CWENR0)	0x02E0	*on page 1116*
Channel Watchdog Enable Register 1 (CWENR1)	0x02E4	*on page 1116*
Channel Watchdog Enable Register 2 (CWENR2)	0x02E8	*on page 1116*
Reserved	0x02EC .. 0x02EF	
Analog Watchdog Out of Range register 0 (AWORR0)	0x02F0	*on page 1118*
Analog Watchdog Out of Range register 1 (AWORR1)	0x02F4	*on page 1118*
Analog Watchdog Out of Range register 2 (AWORR2)	0x02F8	*on page 1118*
Reserved	0x2FC .. 0x02FF	

Table 565 lists the ADC_1 registers with their address offsets and reset values.

Table 565. 12-bit ADC_1 digital registers

Base address: 0xFFE0_4000		
Register name	**Address offset**	**Location**
Main Configuration Register (MCR)	0x0000	*on page 1083*
Main Status Register (MSR)	0x0004	*on page 1086*
Reserved	0x0008 .. 0x000F	
Interrupt Status Register (ISR)	0x0010	*on page 1086*
Channel Pending Register (CEOCFR0)	0x0014	*on page 1087*
Channel Pending Register (CEOCFR1)	0x0018	*on page 1087*
Reserved	0x001C	
Interrupt Mask Register (IMR)	0x0020	*on page 1090*
Channel Interrupt Mask Register (CIMR0)	0x0024	*on page 1090*
Channel Interrupt Mask Register (CIMR1)	0x0028	*on page 1090*
Reserved	0x002C	
Watchdog Threshold Interrupt Status Register (WTISR)	0x0030	*on page 1092*
Watchdog Threshold Interrupt Mask Register (WTIMR)	0x0034	*on page 1093*
Reserved	0x0038 .. 0x003F	
DMA Enable Register (DMAE)	0x0040	*on page 1094*
DMA Channel Select Register 0 (DMAR0)	0x0044	*on page 1095*
DMA Channel Select Register 0 (DMAR1)	0x0048	*on page 1095*
Reserved	0x004C.. 0x005F	
Threshold Register 0 (THRHLR0)	0x0060	*on page 1097*
Threshold Register 1 (THRHLR1)	0x0064	*on page 1097*
Threshold Register 2 (THRHLR2)	0x0068	*on page 1097*
Reserved	0x0070...0x007C	
Presampling Control Register (PSCR)	0x0080	*on page 1099*
Presampling Register 0 (PSR0)	0x0084	*on page 1099*
Presampling Register 1 (PSR1)	0x0088	*on page 1099*
Reserved	0x008C .. 0x0093	
Conversion Timing Register 0 (CTR0)	0x0094	*on page 1101*
Conversion Timing Register 1 (CTR1)	0x0098	*on page 1101*
Reserved	0x009C .. 0x00A3	
Normal Conversion Mask Register 0 (NCMR0)	0x00A4	*on page 1102*
Normal Conversion Mask Register 1 (NCMR1)	0x00A8	*on page 1102*
Reserved	0x00AC .. 0x00B3	
Injected Conversion Mask Register 0 (JCMR0)	0x00B4	*on page 1104*

Table 565. 12-bit ADC_1 digital registers (continued)

Base address: 0xFFE0_4000		
Register name	**Address offset**	**Location**
Injected Conversion Mask Register 1 (JCMR1)	0x00B8	*on page 1104*
Reserved	0x00BC .. 0x00FF	
Channel 0 Data Register (CDR0)	0x0100	*on page 1108*
Channel 1 Data Register (CDR1)	0x0104	*on page 1108*
Channel 2 Data Register (CDR2)	0x0108	*on page 1108*
Channel 3 Data Register (CDR3)	0x010C	*on page 1108*
Channel 4 Data Register (CDR4)	0x0110	*on page 1108*
Channel 5 Data Register (CDR5)	0x0114	*on page 1108*
Channel 6 Data Register (CDR6)	0x0118	*on page 1108*
Channel 7 Data Register (CDR7)	0x011C	*on page 1108*
Channel 8 Data Register (CDR8)	0x0120	*on page 1108*
Channel 9 Data Register (CDR9)	0x0124	*on page 1108*
Channel 10 Data Register (CDR10)	0x0128	*on page 1108*
Channel 11 Data Register (CDR11)	0x012C	*on page 1108*
Channel 12 Data Register (CDR12)	0x0130	*on page 1108*
Channel 13 Data Register (CDR13)	0x0134	*on page 1108*
Channel 14 Data Register (CDR14)	0x0138	*on page 1108*
Channel 15 Data Register (CDR15)	0x013C	*on page 1108*
Reserved	0x0140 .. 0x017F	
Channel 32 Data Register (CDR32)	0x0180	*on page 1108*
Channel 33 Data Register (CDR33)	0x0184	*on page 1108*
Channel 34 Data Register (CDR34)	0x0188	*on page 1108*
Channel 35 Data Register (CDR35)	0x018C	*on page 1108*
Channel 36 Data Register (CDR36)	0x0190	*on page 1108*
Channel 37 Data Register (CDR37)	0x0194	*on page 1108*
Channel 38 Data Register (CDR38)	0x0198	*on page 1108*
Channel 39 Data Register (CDR39)	0x019C	*on page 1108*
Channel 40 Data Register (CDR40)	0x01A0	*on page 1108*
Channel 41 Data Register (CDR41)	0x01A4	*on page 1108*
Channel 42 Data Register (CDR42)	0x01A8	*on page 1108*
Channel 43 Data Register (CDR43)	0x01AC	*on page 1108*
Channel 44 Data Register (CDR44)	0x01B0	*on page 1108*
Reserved	0x01B4 .. 0x02AC	

Table 565. 12-bit ADC_1 digital registers (continued)

Base address: 0xFFE0_4000		
Register name	**Address offset**	**Location**
Channel Watchdog Selection Register 0 (CWSEL0)	0x02B0	*on page 1109*
Channel Watchdog Selection Register 1 (CWSEL1)	0x02B4	*on page 1109*
Reserved	0x02B8 .. 0x02BF	
Channel Watchdog Selection Register 4 (CWSEL4)	0x02C0	*on page 1109*
Channel Watchdog Selection Register 5 (CWSEL5)	0x02C4	*on page 1109*
Reserved	0x02C8 .. 0x02DF	
Channel Watchdog Enable Register 0 (CWENR0)	0x02E0	*on page 1116*
Channel Watchdog Enable Register 1 (CWENR1)	0x02E4	*on page 1116*
Reserved	0x02E8 .. 0x02EF	
Analog Watchdog Out of Range register 0 (AWORR0)	0x02F0	*on page 1118*
Analog Watchdog Out of Range register 0 (AWORR1)	0x02F4	*on page 1118*
Reserved	0x02F8 .. 0x02FF	

32.3.2 Control logic registers

32.3.2.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Offset: 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	EDGLEV	TRGEN	EDGE	XSTRTEN	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	0(1)	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

Figure 665. Main Configuration Register (MCR) – ADC_0

1. This field is reserved but write of 0 is only allowed, value out of reset is zero. Write of 1 to this bit is prohibited.

Offset: 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	0(1)	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

Figure 666. Main Configuration Register (MCR) – ADC_1

1. This field is reserved but write of 0 is only allowed, value out of reset is zero. Write of 1 to this bit is prohibited.

Table 566. Main Configuration Register (MCR) field descriptions

Bit	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)).
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.
EDGLEV(1)	Edge or level selection for external start trigger 0 Edge configuration for external trigger usage. 1 Level configuration for external trigger usage.
TRGEN(1)	External trigger enable. This bit must be set to use external triggering to start a conversion. 0 An external trigger cannot be used to start a conversion. 1 An external trigger can start a conversion.
EDGE(1)	Start trigger edge/ level detection. The following table shows the interaction between the EDGE bit and the TRGEN and EDGLEV bits.

TRGEN	EDGLEV	EDGE	Trigger Detection
0	*n*	*n*	External triggering disabled
1	0	0	External trigger on falling edge of trigger
1	0	1	External trigger on rising edge of trigger
1	1	0	External trigger on low edge of trigger
1	1	1	External trigger on high edge of trigger

Table 566. Main Configuration Register (MCR) field descriptions (continued)

Bit	Description
XSTRTEN(1)	External Start enable If this bit is set, a Normal conversion starts when an external start signal is detected. This can be used to synchronize the start conversion events of two ADCs. 0 START signal is disabled. 1 START signal is asserted when external START is detected.
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.
CTUEN	Cross Trigger Unit Enable 0 CTU triggered conversions disabled 1 CTU triggered conversions enabled
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. 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
OFFREFRESH	Offset phase selection
OFFCANC	Offset phase cancellation selection
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

1. This bit is reserved for ADC_1 as ADC_1 does not have external channel.

32.3.2.2 Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Offset: 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	ACK 0	0	0	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 667. Main Status Register (MSR)

Table 567. Main Status Register (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	This status bit 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 enabled.
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:** WAIT state indicates the delay between switching of external decode (extch channels) signals and the actual start of sampling phase (issue of start pulse). It is used to take into account the settling time of the external mux. This decode delay is programmable by DSD register.

32.3.3 Interrupt registers

32.3.3.1 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

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	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

Figure 668. Interrupt Status Register (ISR)

Table 568. Interrupt Status Register (ISR) field descriptions

Field	Description
EOCTU(1)	End of CTU Conversion interrupt (EOCTU) flag When this bit is set, an EOCTU interrupt has occurred.
JEOC(1)	End of Injected Channel Conversion interrupt (JEOC) flag When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt (JECH) flag When this bit is set, a JECH interrupt has occurred.
EOC(1)	End of Channel Conversion interrupt (EOC) flag When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt (ECH) flag When this bit is set, an ECH interrupt has occurred.

1. Corresponding interrupt can be generated independently by option: 1) eoctu status bit (if enabled by IMR[mskeoctu] bit) or option. 2) channel specific eoc status bits ceocfr0-2[eoc_ch0-95] bit (if enabled by CIMR0-2 registers). Hence both these status bits should be cleared individually on servicing ISR if both the above options are enabled. Preferably only one option should be enabled at a time.

32.3.3.2 Channel Pending Registers (CEOCFR[0..2])

Table 569 shows the exact number of available channels.

Register	Description	ADC
CEOCFR0	End of conversion pending interrupt for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CEOCFR1	End of conversion pending interrupt for channel 32 to 63 (standard channels)	ADC_0
	End of conversion pending interrupt for channel 32 to 44 (standard channels)	ADC_1
CEOCFR2	End of conversion pending interrupt for channel 64 to 95 (external multiplexed channels)	ADC_0

Table 569. CEOCFR[0..2] register description

Offset: 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W																
Reset	0	0	0	0	0	0	0	0	0	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 669. Channel Pending Register 0 (CEOCFR0) — ADC_0 and ADC_1

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	EOC_CH60	EOC_CH59	EOC_CH58	EOC_CH57	EOC_CH56	EOC_CH55	EOC_CH54	EOC_CH53	EOC_CH52	EOC_CH51	EOC_CH50	EOC_CH49	EOC_CH48
W				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_CH47	EOC_CH46	EOC_CH45	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 670. Channel Pending Register 1 (CEOCFR1) – ADC_0

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	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
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 671. Channel Pending Register 1 (CEOCFR1) – ADC_1

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

Figure 672. Channel Pending Register 2 (CEOCFR2) — ADC_0

Table 570. Channel Pending Registers (CEOCFR[0..2]) field descriptions

Field	Description
EOC_CH0	When set, the measure of channel 0 is completed.
EOC_CHn	When set, the measure of channel n is completed.

32.3.3.3 Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

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	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

Figure 673. Interrupt Mask Register (IMR)

Table 571. 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.

32.3.3.4 Channel Interrupt Mask Register (CIMR[0..2])

Table 572 shows the exact number of available channels.

Table 572. CIMR[0..2] register description

Register	Description	ADC
CIMR0	Enable bit for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CIMR1	Enable bit for channel 32 to 63 (standard channels)	ADC_0
	Enable bit for channel 32 to 44 (standard channels)	ADC_1
CIMR2	Enable bit for channel 64 to 95 (external multiplexed channels)	ADC_0

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	CIM15	CIM14	CIM13	CIM12	CIM11	CIM10	CIM9	CIM8	CIM7	CIM6	CIM5	CIM4	CIM3	CIM2	CIM1	CIM0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 674. Channel Interrupt Mask Register 0 (CIMR0) — ADC_0 and ADC_1

Offset: 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIM63	CIM62	CIM61	CIM60	CIM59	CIM58	CIM57	CIM56	CIM55	CIM54	CIM53	CIM52	CIM51	CIM50	CIM49	CIM48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIM47	CIM46	CIM45	CIM44	CIM43	CIM42	CIM41	CIM40	CIM39	CIM38	CIM37	CIM36	CIM35	CIM34	CIM33	CIM32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 675. Channel Interrupt Mask Register 1 (CIMR1) – ADC_0

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	CIM44	CIM43	CIM42	CIM41	CIM40	CIM39	CIM38	CIM37	CIM36	CIM35	CIM34	CIM33	CIM32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 676. Channel Interrupt Mask Register 1 (CIMR1) – ADC_1

Offset: 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIM95	CIM94	CIM93	CIM92	CIM91	CIM90	CIM89	CIM88	CIM87	CIM86	CIM85	CIM84	CIM83	CIM82	CIM81	CIM80
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIM79	CIM78	CIM77	CIM76	CIM75	CIM74	CIM73	CIM72	CIM71	CIM70	CIM69	CIM68	CIM67	CIM66	CIM65	CIM64
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 677. Channel Interrupt Mask Register 2 (CIMR2) — ADC_0

Table 573. Channel Interrupt Mask Register (CIMR[0..2]) field descriptions

Field	Description
CIM0	Interrupt enable When set (CIM0 = 1), interrupt for channel 0 is enabled.
CIMn	Interrupt enable When set (CIMn = 1), interrupt for channel n is enabled.

32.3.3.5 Watchdog Threshold Interrupt Status Register (WTISR)

For ADC_0 (10-bit)

Reset value: 0x0000_0000

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	0	0	0	WDG5H	WDG5L	WDG4H	WDG4L	WDG3H	WDG3L	WDG2H	WDG2L	WDG1H	WDG1L	WDG0H	WDG0L
W					w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 678. Watchdog Threshold Interrupt Status Register (WTISR) – ADC_0

For ADC_1 (12-bit)

Reset value: 0x0000_0000

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	0	0	0	0	0	0	0	0	0	WDG2H	WDG2L	WDG1H	WDG1L	WDG0H	WDG0L
W											w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 679. Watchdog Threshold Interrupt Status Register (WTISR) – ADC_1

Table 574. Watchdog Threshold Interrupt Status Register (WTISR) field descriptions – ADC_1

Field	Description
WDGxH [x = 0..2]	This corresponds to the interrupt generated on the converted value being higher than the programmed higher threshold.
WDGxL [x = 0..2]	This corresponds to the interrupt generated on the converted value being lower than the programmed lower threshold.

32.3.3.6 Watchdog Threshold Interrupt Mask Register (WTIMR)

For ADC_0 (10-bit):

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	0	0	0	0	MSKWDG5H	MSKWDG5L	MSKWDG4H	MSKWDG4L	MSKWDG3H	MSKWDG3L	MSKWDG2H	MSKWDG2L	MSKWDG1H	MSKWDG1L	MSKWDG0H	MSKWDG0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 680. Watchdog Threshold Interrupt Mask Register (WTIMR) – ADC_0

Table 575. Watchdog Threshold Interrupt Mask Register (WTIMR) field descriptions – ADC_0

Field	Description
MSKWDGxH [x = 0..5]	This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold. When set the interrupt is enabled.
MSKWDGxL [x = 0..5]	This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold. When set the interrupt is enabled.

For ADC_1 (12-bit)

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	0	0	0	0	0	0	0	0	0	0	MSKWDG2H	MSKWDG2L	MSKWDG1H	MSKWDG1L	MSKWDG0H	MSKWDG0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 681. Watchdog Threshold Interrupt Mask Register (WTIMR) – ADC_1

Table 576. Watchdog Threshold Interrupt Mask Register (WTIMR) field descriptions – ADC_1

Field	Description
MSKWDGxH [x = 0..2]	This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold. When set the interrupt is enabled.
MSKWDGxL [x = 0..2]	This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold. When set the interrupt is enabled.

32.3.4 DMA registers

32.3.4.1 DMA Enable Register (DMAE)

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

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	0	0	0	0	0	0	DCLR	DMAEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 682. DMA Enable Register (DMAE)

Table 577. DMA Enable Register (DMAE) field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request cleared by Acknowledge from DMA controller 1 DMA request cleared on read of data registers
DMAEN	DMA global enable 0 DMA feature disabled 1 DMA feature enabled

32.3.4.2 DMA Channel Select Register (DMAR[0..2])

Table 578 shows the exact number of available channels.

Register	Description	ADC
DMAR0	Enable bits for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
DMAR1	Enable bits for channel 32 to 63 (standard channels)	ADC_0
	Enable bits for channel 32 to 44 (standard channels)	ADC_1
DMAR2	Enable bits for channel 64 to 95 (external multiplexed channels)	ADC_0

Table 578. DMAR[0..2] register description

Offset: 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	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	DMA15	DMA14	DMA13	DMA12	DMA11	DMA10	DMA9	DMA8	DMA7	DMA6	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 683. DMA Channel Select Register 0 (DMAR0) — ADC_0 and ADC_1

Offset: 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DMA63	DMA62	DMA61	DMA60	DMA59	DMA58	DMA57	DMA56	DMA55	DMA54	DMA53	DMA52	DMA51	DMA50	DMA49	DMA48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA47	DMA46	DMA45	DMA44	DMA43	DMA42	DMA41	DMA40	DMA39	DMA38	DMA37	DMA36	DMA35	DMA34	DMA33	DMA32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 684. DMA Channel Select Register 1 (DMAR1) – ADC_0

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	DMA44	DMA43	DMA42	DMA41	DMA40	DMA39	DMA38	DMA37	DMA36	DMA35	DMA34	DMA33	DMA32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 685. DMA Channel Select Register 1 (DMAR1) – ADC_1

Offset: 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DMA95	DMA4	DMA93	DMA92	DMA91	DMA90	DMA89	DMA88	DMA87	DMA86	DMA85	DMA84	DMA83	DMA82	DMA81	DMA80
Reset	0	0	0	0	0	0	0	0	0	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	DMA79	DMA78	DMA77	DMA76	DMA75	DMA74	DMA73	DMA72	DMA71	DMA70	DMA69	DMA68	DMA67	DMA66	DMA65	DMA64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 686. DMA Channel Select Register 2 (DMAR2) — ADC_0

Table 579. DMA Channel Select Register (DMAR[0..2]) field descriptions

Field	Description
31	DMA0: DMA enable When set (DMA0 = 1), channel 0 is enabled to transfer data in DMA mode.
n	DMAn: DMA enable When set (DMAn = 1), channel n is enabled to transfer data in DMA mode.

32.3.5 Threshold Register

The six THRHLR*n* registers are used to store the user-programmable thresholds' values.

For 10-bit ADC:

Offset: 0x0060 (THRHLR0)
Offset: 0x0064 (THRHLR1)
Offset: 0x0068 (THRHLR2)
Offset: 0x006C (THRHLR3)
Offset: 0x0280 (THRHLR4)
Offset:0x0284 (THRHLR5)

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

Figure 687. Threshold Register – ADC_0

Table 580. Threshold Register field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

Table 581. ADC_0 Threshold Register (THRHLR) field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

For ADC_1 (12-bit): THRHLR[0..2]

Offset: 0x0060 (THRHLR0)
Offset: 0x0064 (THRHLR1)
Offset: 0x0068 (THRHLR2)

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	THRH											
W																
Reset	0	0	0	0	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	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 688. ADC_1 Threshold Register THRHLR – ADC_1

Table 582. ADC_1 Threshold Register (THRHLR) field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

32.3.6 Presampling registers

32.3.6.1 Presampling Control Register (PSCR)

Offset: 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		PRECONV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 689. Presampling Control Register (PSCR)

Table 583. Presampling Control Register (PSCR) field descriptions

Field	Description
PREVAL2	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (external channels). See *Table 585*.
PREVAL1	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (internal extended channels).
PREVAL0	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (internal precision channels).
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.

32.3.6.2 Presampling Register (PSR[0..2])

Table 584 shows the exact number of available channels.

Table 584. PSR[0..2] register description

Register	Description	ADC
PSR0	Enable bits of presampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
PSR1	Enable bits of presampling for channel 32 to 63 (standard channels)	ADC_0
	Enable bits of presampling for channel 32 to 44 (standard channels)	ADC_1
PSR2	Enable bits of presampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Offset: 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	PRES15	PRES14	PRES13	PRES12	PRES11	PRES10	PRES9	PRES8	PRES7	PRES6	PRES5	PRES4	PRES3	PRES2	PRES1	PRES0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 690. Presampling Register 0 (PSR0) — ADC_0 and ADC_1

Offset: 0x0088 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRES63	PRES62	PRES61	PRES60	PRES59	PRES58	PRES57	PRES56	PRES55	PRES54	PRES53	PRES52	PRES51	PRES50	PRES49	PRES48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRES47	PRES46	PRES45	PRES44	PRES43	PRES42	PRES41	PRES40	PRES39	PRES38	PRES37	PRES36	PRES35	PRES34	PRES33	PRES32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 691. Presampling Register 1 (PSR1) – ADC_0

Offset: 0x0088 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R				PRES44	PRES43	PRES42	PRES41	PRES40	PRES39	PRES38	PRES37	PRES36	PRES35	PRES34	PRES33	PRES32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 692. Presampling Register 1 (PSR1) – ADC_1

Offset: 0x008C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRES95	PRES94	PRES93	PRES92	PRES91	PRES90	PRES89	PRES88	PRES87	PRES86	PRES85	PRES84	PRES83	PRES82	PRES81	PRES80
Reset	0	0	0	0	0	0	0	0	0	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	PRES79	PRES78	PRES77	PRES76	PRES75	PRES74	PRES73	PRES72	PRES71	PRES70	PRES69	PRES68	PRES67	PRES66	PRES65	PRES64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 693. Presampling Register 2 (PSR2) — ADC_0

Table 585. Presampling Register (PSR[0..2]) field descriptions

Field	Description
PRES0	Presampling enable When set (PRES0 = 1), presampling is enabled for channel 0.
PRESn	Presampling enable When set (PRESn = 1), presampling is enabled for channel n.

32.3.6.3 Conversion timing register

Table 586 shows the exact number of available channels.

Table 586. CTR[0..2] register description

Register	Description	ADC
CTR0	Associated to internal precision channels (from 0 to 15)	ADC_0
		ADC_1
CTR1	Associated to internal standard channels (from 32 to 63)	ADC_0
	Associated to internal standard channels (from 32 to 44)	ADC_1
CTR2	Associated to external multiplexed channels (from 64 to 95)	ADC_0

Offset: 0x0094 (CTR0) — ADC_0 and ADC_1
Offset: 0x0098 (CTR1) — ADC_0 and ADC_1
Offset: 0x009C (CTR2) — ADC_0

Access: User read/write

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

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

Figure 694. Conversion timing register

Table 587. Conversion timing register 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
INPCMP	Configuration bits for comparison phase duration 00 4 ADC clock cycles 01 1 ADC clock cycle 10 2 ADC clock cycles 11 3 ADC clock cycles
INPSAMP	Configuration bits for sampling phase duration

32.3.7 Mask registers

32.3.7.1 Introduction

These registers are used to program which of the 96 input channels must be converted during Normal and Injected conversion.

32.3.7.2 Normal Conversion Mask Registers (NCMR[0..2])

Table 588 shows the exact number of available channels.

Table 588. NCMR[0..2] register description

Register	Description	ADC
NCMR0	Enable bits of normal sampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
NCMR1	Enable bits of normal sampling for channel 32 to 63 (standard channels)	ADC_0
	Enable bits of normal sampling for channel 32 to 44 (standard channels)	ADC_1
NCMR2	Enable bits of normal sampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Note: The implicit channel conversion priority in the case in which all channels are selected is the following: ADCn_P[0:x], ADCn_S[0:y], ADCn_X[0:z].The channels always start with 0, the lowest index.

Offset: 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	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 695. Normal Conversion Mask Register 0 (NCMR0) — ADC_0 and ADC_1

Offset: 0x00A8: Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CH63	CH62	CH61	CH60	CH59	CH58	CH57	CH56	CH55	CH54	CH53	CH52	CH51	CH50	CH49	CH48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 696. Normal Conversion Mask Register 1 (NCMR1) – ADC_0

Offset: 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	0	0	0	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 697. Normal Conversion Mask Register 1 (NCMR1) – ADC_1

Offset: 0x00AC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 698. Normal Conversion Mask Register 2 (NCMR2) — ADC_0

Table 589. Normal Conversion Mask Registers (NCMR[0..2]) field descriptions

Field	Description
CH0	Sampling enable When set Sampling is enabled for channel 0.
CHn	Sampling enable When set Sampling is enabled for channel n.

32.3.7.3 Injected Conversion Mask Registers (JCMR[0..2])

Table 590 shows the exact number of available channels.

Table 590. JCMR[0..2] register description

Register	Description	ADC
JCMR0	Enable bits of injected sampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
JCMR1	Enable bits of injected sampling for channel 32 to 63 (standard channels)	ADC_0
	Enable bits of injected sampling for channel 32 to 44 (standard channels)	ADC_1
JCMR2	Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Offset: 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	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 699. Injected Conversion Mask Register 0 (JCMR0) — ADC_0 and ADC_1

Offset 0x00B8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CH63	CH62	CH61	CH60	CH59	CH58	CH57	CH56	CH55	CH54	CH53	CH52	CH51	CH50	CH49	CH48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 700. Injected Conversion Mask Register 1 (JCMR1) – ADC_0

Offset : 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	0	0	0	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 701. Injected Conversion Mask Register 1 (JCMR1) — ADC_1

Offset: 0x00BC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 702. Injected Conversion Mask Register 2 (JCMR2) — ADC_0

Table 591. Injected Conversion Mask Registers (JCMR[0..2]) field descriptions

Field	Description
CH0	Sampling enable When set, sampling is enabled for channel 0.
CHn	Sampling enable When set, sampling is enabled for channel n.

32.3.8 Delay registers

32.3.8.1 Decode Signals Delay Register (DSDR)

DSDR is implemented in ADC_0 only.

Offset: 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	DSD											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 703. Decode Signals Delay Register (DSDR)

Table 592. 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: ADC_DSDR = 0; 0 ADC clock cycle delay ADC_DSDR = 2; 1 ADC clock cycle delay ADC_DSDR = 4; 2 ADC clock cycles delay

32.3.8.2 Power-down Exit Delay Register (PDEDR)

PDEDR is implemented in ADC_0 only.

Offset: 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

Figure 704. Power-down Exit Delay Register (PDEDR)

Table 593. Power-down Exit Delay Register (PDEDR) field descriptions

Field	Description
PDED	Delay between the power-down bit reset and the start of conversion The power down delay is calculated as: PDED x 1/frequency of ADC clock.

32.3.9 Data registers

32.3.9.1 Introduction

Table 594 shows the exact number of available channels.

Table 594. CDR[0..95] register description

Register	Description	ADC
CDR0	Enable bits of injected sampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1

Table 594. CDR[0..95] register description

Register	Description	ADC
CDR1	Enable bits of injected sampling for channel 32 to 63 (standard channels)	ADC_0
	Enable bits of injected sampling for channel 32 to 44 (standard channels)	ADC_1
CDR2	Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels)	ADC_0

32.3.9.2 Channel Data Register (CDR[0..95])

For ADC_0 10-bit:

Address: See *Table 564* 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									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 705. Channel Data Register (CDR[0..95]) — ADC_0

Table 595. Channel Data Register (CDR[0..95]) field descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]: – When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read. – When OWREN = 1, then OVERW flags the CDATA field overwrite status. 0 Converted data has not been overwritten 1 Previous converted data has been overwritten before having been read
RESULT	This bit reflects the mode of conversion for the corresponding channel. 00 Data is a result of Normal conversion mode 01 Data is a result of Injected conversion mode 10 Data is a result of CTU conversion mode 11 Reserved
CDATA	Channel 0-95 converted data

For ADC_1 12-bit:

Address: See *Table 564* 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	CDATA											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 706. Channel Data Register (CDR[0..95]) — ADC_1

32.3.9.3 Channel Watchdog Select Register (CWSELR[0..11])

Register WSEL_CHn[3:0] = Selects the threshold register which provides the values to be used for upper and lower bounds for channel n. *Table 596* shows the exact number of available channels.

Table 596. CWSELR[0..11] register description

Register	Description	ADC
CWSELR[0..1]	Channel watchdog select register for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CWSELR[4..7]	Channel watchdog select register for channel 32 to 63 (standard channels)	ADC_0
CWSELR[4..5]	Channel watchdog select register for channel 32 to 44 (standard channels)	ADC_1
CWSELR[8..11]	Channel watchdog select register for channel 64 to 95 (external multiplexed channels)	ADC_0

Offset: 0x02B0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH7			0	WSEL_CH6			0	WSEL_CH5			0	WSEL_CH4		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH3			0	WSEL_CH2			0	WSEL_CH1			0	WSEL_CH0		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 707. Channel Watchdog Select Register 0 (CWSELR0) – ADC_0

Table 597. CWSELR field descriptions – ADC_0

Field	Description
WSEL_CHn	Channel Watchdog select for channel n 000: THRHLR0 register is selected 001: THRHLR1 register is selected 010: THRHLR2 register is selected 011: THRHLR3 register is selected 100: THRHLR4 register is selected 101: THRHLR5 register is selected 110: RESERVED 111: RESERVED

Offset: 0x02B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH15			0	WSEL_CH14			0	WSEL_CH13			0	WSEL_CH12		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH11			0	WSEL_CH10			0	WSEL_CH9			0	WSEL_CH8		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 708. Channel Watchdog Select Register 1 (CWSELR1) – ADC_0

Table 598. CWSELR1 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH39			0	WSEL_CH38			0	WSEL_CH37			0	WSEL_CH36		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH35			0	WSEL_CH34			0	WSEL_CH33			0	WSEL_CH32		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 709. Channel Watchdog Select Register 4 (CWSELR4) – ADC_0

Table 599. CWSELR4 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH47			0	WSEL_CH46			0	WSEL_CH45			0	WSEL_CH44		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH43			0	WSEL_CH42			0	WSEL_CH41			0	WSEL_CH40		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 710. Channel Watchdog Select Register 5 (CWSELR5) – ADC_0

Table 600. CWSELR5 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597.*

Offset: 0x02C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH55			0	WSEL_CH54			0	WSEL_CH53			0	WSEL_CH52		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH51			0	WSEL_CH50			0	WSEL_CH49			0	WSEL_CH48		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 711. Channel Watchdog Select Register 6 (CWSELR6) – ADC_0

Table 601. CWSELR6 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02CC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH63			0	WSEL_CH62			0	WSEL_CH61			0	WSEL_CH60		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH59			0	WSEL_CH58			0	WSEL_CH57			0	WSEL_CH56		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 712. Channel Watchdog Select Register 7 (CWSELR7) – ADC_0

Table 602. CWSELR7 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02D0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH71			0	WSEL_CH70			0	WSEL_CH69			0	WSEL_CH68		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH67			0	WSEL_CH66			0	WSEL_CH65			0	WSEL_CH64		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 713. Channel Watchdog Select Register 8 (CWSELR8) – ADC_0

Table 603. CWSELR field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02D4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH79			0	WSEL_CH78			0	WSEL_CH77			0	WSEL_CH76		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH75			0	WSEL_CH74			0	WSEL_CH73			0	WSEL_CH72		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 714. Channel Watchdog Select Register 9 (CWSELR9) – ADC_0

Table 604. CWSELR9 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02D8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH87			0	WSEL_CH86			0	WSEL_CH85			0	WSEL_CH84		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH83			0	WSEL_CH82			0	WSEL_CH81			0	WSEL_CH80		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 715. Channel Watchdog Select Register 10 (CWSELR10) – ADC_0

Table 605. CWSELR10 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02DC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	WSEL_CH95			0	WSEL_CH94			0	WSEL_CH93			0	WSEL_CH92		
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH91			0	WSEL_CH90			0	WSEL_CH89			0	WSEL_CH88		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 716. Channel Watchdog Select Register 11 (CWSELR11) – ADC_0

Table 606. CWSELR11 field descriptions – ADC_0

Field	Description
WSEL_CHn	See *Table 597*.

ADC_1

Offset: 0x02B0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH7		0	0	WSEL_CH6		0	0	WSEL_CH5		0	0	WSEL_CH4	
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH3		0	0	WSEL_CH2		0	0	WSEL_CH1		0	0	WSEL_CH0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 717. Channel Watchdog Select Register 0 (CWSELR0) – ADC_1

Table 607. CWSELR0 field descriptions – ADC_1

Field	Description
WSEL_CHn	Channel Watchdog select for channel n 00: THRHLR0 register is selected 01: THRHLR1 register is selected 10: THRHLR2 register is selected 11: RESERVED

Offset: 0x02B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH15		0	0	WSEL_CH14		0	0	WSEL_CH13		0	0	WSEL_CH12	
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH11		0	0	WSEL_CH10		0	0	WSEL_CH9		0	0	WSEL_CH8	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 718. Channel Watchdog Select Register 1 (CWSELR1) – ADC_1

Table 608. CWSELR1 field descriptions – ADC_1

Field	Description
WSEL_CHn:	See *Table 597*.

Offset: 0x02C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH39		0	0	WSEL_CH38		0	0	WSEL_CH37		0	0	WSEL_CH36	
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH35		0	0	WSEL_CH34		0	0	WSEL_CH33		0	0	WSEL_CH32	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 719. Channel Watchdog Select Register 4 (CWSELR4) – ADC_1

Table 609. CWSELR4 field descriptions – ADC_1

Field	Description
WSEL_CHn	See *Table 597*.

Offset: 0x02C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH47		0	0	WSEL_CH46		0	0	WSEL_CH45		0	0	WSEL_CH44	
W																
Reset	0	0	0	0	0	0	0	0	0	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	WSEL_CH43		0	0	WSEL_CH42		0	0	WSEL_CH41		0	0	WSEL_CH40	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 720. Channel Watchdog Select Register 5 (CWSELR5) – ADC_1

Table 610. CWSELR5 field descriptions – ADC_1

Field	Description
WSEL_CHn	See *Table 597*.

32.3.9.4 Channel Watchdog Enable Register (CWENRx, x = [0..2])

Table 611 shows the exact number of available channels.

Table 611. CWENR[0..2] register description

Register	Description	ADC
CWENR0	Watchdog enable bits for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CWENR1	Watchdog enable bits for channel 32 to 63 (standard channels)	ADC_0
	Watchdog enable bits for channel 32 to 44 (standard channels)	ADC_1
CWENR2	Watchdog enable bits for channel 64 to 95 (external multiplexed channels)	ADC_0

Offset: 0x02E0 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	CWEN15	CWEN14	CWEN13	CWEN12	CWEN11	CWEN10	CWEN9	CWEN8	CWEN7	CWEN6	CWEN5	CWEN4	CWEN3	CWEN2	CWEN1	CWEN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 721. Channel Watchdog Enable Register 0 (CWENR0) — ADC_0 and ADC_1

Offset: 0x02E4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CWEN63	CWEN62	CWEN61	CWEN60	CWEN59	CWEN58	CWEN57	CWEN56	CWEN55	CWEN54	CWEN53	CWEN52	CWEN51	CWEN50	CWEN49	CWEN48
Reset	0	0	0	0	0	0	0	0	0	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	CWEN47	CWEN46	CWEN45	CWEN44	CWEN43	CWEN42	CWEN41	CWEN40	CWEN39	CWEN38	CWEN37	CWEN36	CWEN35	CWEN34	CWEN33	CWEN32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 722. Channel Watchdog Enable Register 1 (CWENR1) – ADC_0

Offset: 0x02E4 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	CWEN44	CWEN43	CWEN42	CWEN41	CWEN40	CWEN39	CWEN38	CWEN37	CWEN36	CWEN35	WEN334	CWEN33	CWEN32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 723. Channel Watchdog Enable Register 1 (CWENR1) – ADC_1

Offset: 0x02E8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CWEN95	CWEN94	CWEN93	CWEN92	CWEN91	CWEN90	CWEN89	CWEN88	CWEN87	CWEN86	CWEN85	CWEN84	CWEN83	CWEN82	CWEN81	CWEN80
Reset	0	0	0	0	0	0	0	0	0	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	CWEN79	CWEN78	CWEN77	CWEN76	CWEN75	CWEN74	CWEN73	CWEN72	CWEN71	CWEN70	CWEN69	CWEN68	CWEN67	CWEN66	CWEN65	CWEN64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 724. Channel Watchdog Enable Register 2 (CWENR2) — ADC_0

Table 612. Channel Watchdog Enable Register (CWENRx, x = [0..2]) field descriptions

Field	Description
CWENn	Channel Watchdog enable When set (CWENn = 1) Watchdog feature is enabled for channel n.

32.3.9.5 Analog Watchdog Out of Range Register (AWORRx, x = [0..2])

Table 613. Analog Watchdog Out of Range Register 0 (AWORR0) — ADC_0 and ADC_1 field description

Register	Description	ADC
AWORR0	Analog watchdog out of range register for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
AWORR1	Analog watchdog out of range register for channel 32 to 63 (standard channels)	ADC_0
	Analog watchdog out of range register for channel 32 to 44 (standard channels)	ADC_1
AWORR2	Analog watchdog out of range register for channel 64 to 95 (external multiplexed channels)	ADC_0

Offset: 0x02F0 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	AWOR_CH15	AWOR_CH14	AWOR_CH13	AWOR_CH12	AWOR_CH11	AWOR_CH10	AWOR_CH9	AWOR_CH8	AWOR_CH7	AWOR_CH6	AWOR_CH5	AWOR_CH4	AWOR_CH3	AWOR_CH2	AWOR_CH1	AWOR_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 725. Analog Watchdog Out of Range Register 0 (AWORR0) — ADC_0 and ADC_1

Offset: 0x02F4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWOR_CH63	AWOR_CH62	AWOR_CH61	AWOR_CH60	AWOR_CH59	AWOR_CH58	AWOR_CH57	AWOR_CH56	AWOR_CH55	AWOR_CH54	AWOR_CH53	AWOR_CH52	AWOR_CH51	AWOR_CH50	AWOR_CH49	AWOR_CH48
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	AWOR_CH47	AWOR_CH46	AWOR_CH45	AWOR_CH44	AWOR_CH43	AWOR_CH42	AWOR_CH41	AWOR_CH40	AWOR_CH39	AWOR_CH38	AWOR_CH37	AWOR_CH36	AWOR_CH35	AWOR_CH34	AWOR_CH33	AWOR_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 726. Analog Watchdog Out of Range Register 1 (AWORR1) – ADC_0

Offset: 0x02F4 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	AWOR_CH44	AWOR_CH43	AWOR_CH42	AWOR_CH41	AWOR_CH40	AWOR_CH39	AWOR_CH38	AWOR_CH37	AWOR_CH36	AWOR_CH35	AWOR_CH34	AWOR_CH33	AWOR_CH32
W				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 727. Analog Watchdog Out of Range Register 1 (AWORR1) – ADC_1

Offset: 0x02F8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWOR_CH95	AWOR_CH94	AWOR_CH93	AWOR_CH92	AWOR_CH91	AWOR_CH90	AWOR_CH89	AWOR_CH88	AWOR_CH87	AWOR_CH86	AWOR_CH85	AWOR_CH84	AWOR_CH83	AWOR_CH82	AWOR_CH81	AWOR_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	AWOR_CH79	AWOR_CH78	AWOR_CH77	AWOR_CH76	AWOR_CH75	AWOR_CH74	AWOR_CH73	AWOR_CH72	AWOR_CH71	AWOR_CH70	AWOR_CH69	AWOR_CH68	AWOR_CH67	AWOR_CH66	AWOR_CH65	AWOR_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

Figure 728. Analog Watchdog Out of Range Register 2 (AWORR2) — ADC_0

32.4 Functional description

32.4.1 Analog channel conversion

ADC digital interface supports three conversion modes:

- Normal conversion
- Injected conversion
- CTU triggered conversion

32.4.1.1 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).

32.4.1.2 Start of normal conversion

The conversion chain starts when the NSTART bit in the *Main Configuration Register (MCR) – ADC_0* is set.

By programming the configuration bits in the *Main Configuration Register (MCR) – ADC_0*, the normal conversion can be started in two ways:

- By software – The conversion chain starts when the NSTART bit in the MCR is set.
- By trigger – An on-chip internal signal triggers an ADC conversion. The settings in the MCR select how conversions are triggered based on these internal signals:
  - If the EDGLEV (edge/level selection) bit in the MCR is cleared, then a rising/falling edge (depending on the EDGE bit in MCR) detected in the signal sets the

NSTART bit in the MSR 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 NSTART bit in the MCR is set and the programmed level on the trigger signal is detected. The level is selected using the EDGE bit in the MCR. 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 NSTART status bit in the MSR is automatically set when the normal conversion starts. At the same time the NSTART bit in the MCR 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 is immediately issued after the start of conversion.

32.4.1.3 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 MODE bit in the MCR. 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 729*.

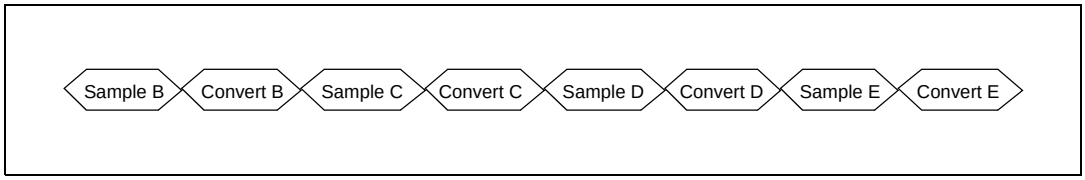


Figure 729. Normal conversion flow

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 1 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 NSTART bit in the MCR 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 NSTART status bit in the MSR is automatically set when the Normal conversion starts. Unlike One Shot Mode, the NSTART bit in the MCR 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 NSTART bit in the MSR. But, if NSTART is reset during last channel conversion of current chain then one more chain would execute before conversion actually stops.

Example 2 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 NSTART bit in the MCR is cleared by software.

If the conversion is started by an external trigger and EDGLEV is ‘0’, the NSTART bit in the MCR 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).

32.4.1.4 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 730*.

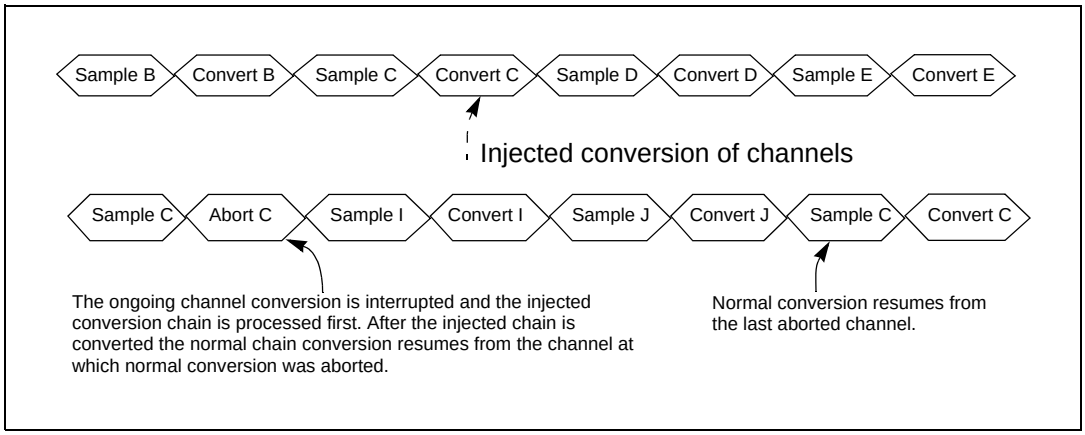


Figure 730. Injected sample/conversion sequence

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, setting the MCR[JTRGEN]; a programmed event (rising/falling edge depending on MCR[JTRGEN]) on the signal coming from PIT_RTI or CTU starts the injected conversion by setting the JSTART bit in the MSR. At the end of the chain, the JSTART bit in the MSR is cleared and the normal conversion chain is resumed.

The JSTART status bit in the MSR is automatically set when the Injected conversion starts. At the same time the JSTART bit in the MCR 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 IMR[MSKJEOC]) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by IMR[MSKJECH]).

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.

32.4.1.5 Abort conversion

Two different abort functions are provided.

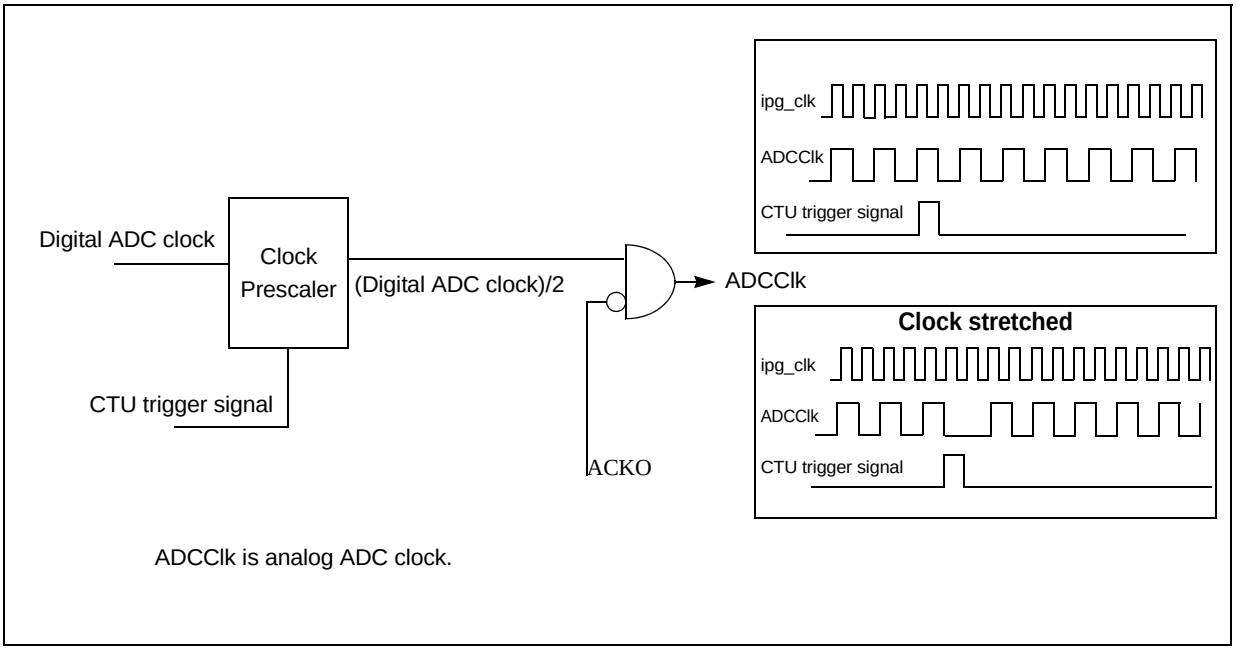
- The user can abort the ongoing conversion by setting the ABORT bit in the MCR. The current conversion is aborted and the conversion of the next channel of the chain is immediately started. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is reset after the conversion 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).

Note: ABORT or ABORTCHAIN should be set only when the conversion is actually ongoing i.e. when either of the MSR[NSTART or JSTART or CTUSTART] status bits is set.

32.4.2 Analog clock generator and conversion timings

The analog ADC clock is always half of the digital ADC clock. SPC564Bxx and SPC56ECxx supports clock stretching as shown in *Figure 731*.

Figure 731. Prescaler simplified block diagram



32.4.3 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. For only the 10-bit ADC 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 INPSAMPLE 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}).

32.4.3.1 ADC_0 sampling and conversion timing

The sampling phase duration of ADC_0 is

$$T_{sample} = (INPSAMPLE - ndelay) \bullet T_{ck}$$

$$INPSAMPLE \geq 3$$

where ndelay is equal to 0.5 if INPSAMPLE is less than or equal to 06h, otherwise it is 1. INPSAMPLE must be greater than or equal to 3 (hardware requirement). See *Figure 694*.

The total evaluation phase duration for ADC_0 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 614. 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

32.4.3.2 ADC_1 sampling and conversion timing

The sampling phase duration is:

$$T_{sample} = (INPSAMPLE - 1) \bullet T_{ck}$$

$$INPSAMPLE \geq 8$$

The total evaluation phase duration for ADC_1 is:

$$T_{eval} = 12 \bullet T_{biteval} = 12 \bullet (INPCMP \bullet T_{ck})$$

$$(INPCMP \geq 1) \quad \text{and} \quad (INPLATCH < INPCMP)$$

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):

$$T_{conv} = T_{sample} + T_{eval} + T_{ck}$$

The timings refer to the unit T_{ck}, where f_{ck} = (1/2 x ADC peripheral set clock).

32.4.4 ADC CTU (Cross Triggering Unit)

32.4.4.1 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_RTI) 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.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and ADC are synchronous with the peripheral set 3 clock in both cases.

32.4.4.2 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_RTI 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.

Note: If CTU trigger arrives in the evolution phase of the ongoing conversion, the conversion after the ctu channel will be the next queued normal conversion channel.

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.

However, it is recommended not to initiate software injected conversion while in CTU trigger mode.

32.4.5 Presampling

32.4.5.1 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, 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 732*, *Figure 733*).

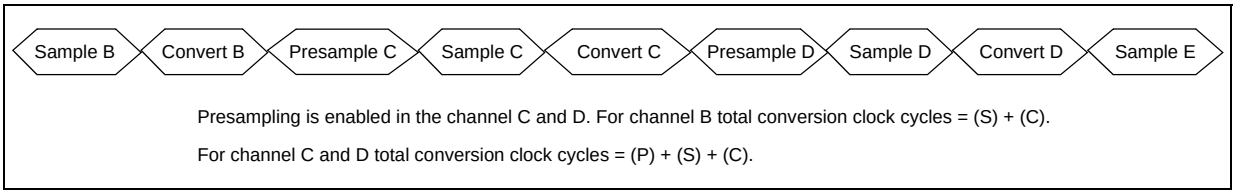


Figure 732. Presampling sequence

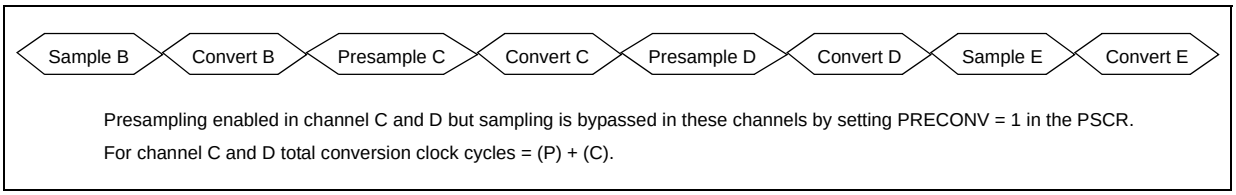


Figure 733. Presampling sequence with PRECONV = 1

32.4.5.2 Presampling channel enable signals

It is possible to select between two internally generated voltages V0 and V1 depending on the value of the PREVAL fields in the PSCR as shown in *Table 615*.

Table 615. Presampling voltage selection based on PREVALx fields

PSCR[PREVALx]	Presampling voltage
00	V0 = $V_{SS_HV_ADC0}$ or $V_{SS_HV_ADC1}$
01	V1 = $V_{DD_HV_ADC0}$ or $V_{DD_HV_ADC1}$
10	Reserved
11	Reserved

Several presampling value fields, one per channel type, in the PSCR make it possible to select different presampling values for each type.

32.4.6 Programmable analog watchdog

32.4.6.1 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 734*) specified by an upper and a lower threshold value named THRH and THRL respectively.

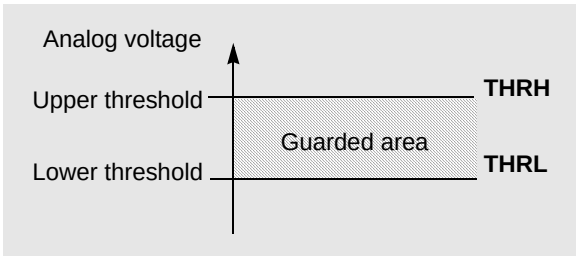


Figure 734. Guarded area

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 616*. Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

Table 616. 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

Each channel can be enabled independently from the CWENR registers and can select the watchdog threshold registers (THRHLRx) to be used by programming the CWSELR registers. The threshold registers selected by the WSEL_CHx field of the CWSELR will provide the threshold values.

For example, if channel number 15 is to be monitored with the threshold values in THRHLR1, then WSEL_CH15 in the CWSELR is programmed to select THRHLR1 to provide the threshold values. The channel monitoring is enabled by setting the bit corresponding to channel 15 in the CWENR.

If a converted value for a particular channel lies outside the range specified by threshold values, then the corresponding bit is set in the Analog Watchdog Out of Range Register (AWORR).

A set of threshold registers (THRHLRx) can be linked to several ADC channels. The threshold values to be selected for a channel need be programmed only once in the CWSELRx.

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.

32.4.7 DMA functionality

A DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR registers. The DMAR masking registers must be programmed before starting any conversion. There is one DMAR per channel type and each ADC module has one DMA request associated with it.

The DMA transfers can be enabled using the DMAEN bit of DMAE register. When the DCLR bit of DMAE register is set then the DMA request is cleared on the reading of the register for which DMA transfer has been enabled.

32.4.8 Interrupts

The ADC generates the following two 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. Two 7-bit registers named CEOCFR (Channel Pending Registers) and IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt request to INTC module.

Interrupts can be individually enabled on a channel by channel base by programming the CIMR (Channel Interrupt Mask Register).

Several *Channel Pending Register 0 (CEOCFR0) — ADC_0 and ADC_1* are also provided 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 the corresponding pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored.

The CEOCFR 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 must be maintained at '0').

32.4.9 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[0:11].

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.

32.4.10 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).

32.4.11 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 ACKO bit in the MCR. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

33 Cross Triggering Unit (CTU)

33.1 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_RTI. 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.

33.2 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_RTI for clearing the flag
- Synchronization with ADC to avoid collision

33.3 Block diagram

The CTU block diagram is shown in *Figure 735*.

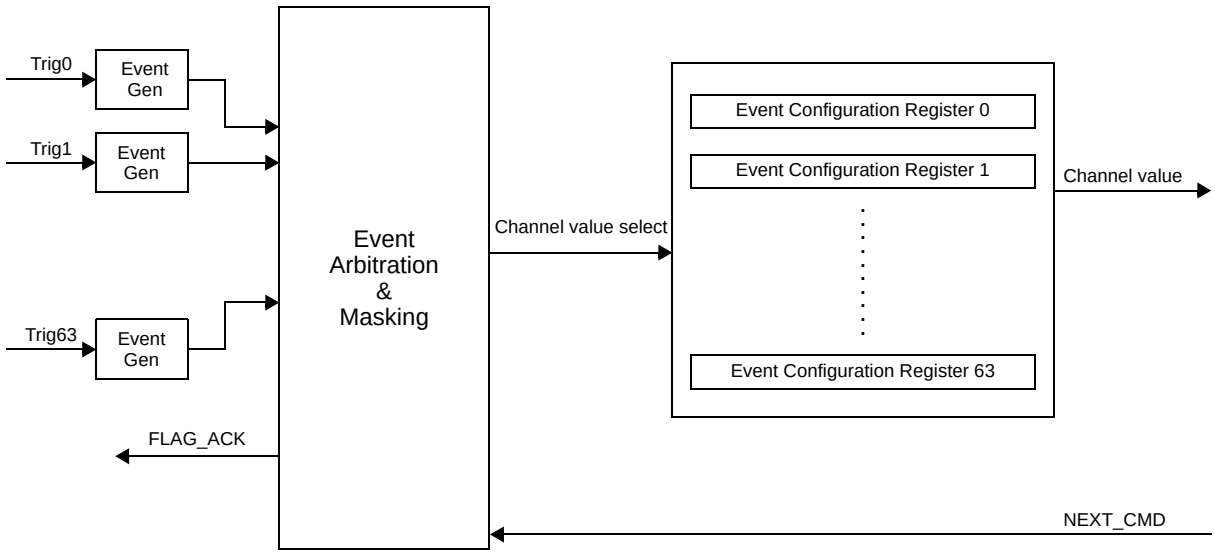


Figure 735. Cross Triggering Unit block diagram

33.4 Memory map and register descriptions

The CTU registers are listed in *Table 617*. Every register can have 32-bit access. The base address of the CTU is 0xFFE6_4000.

Table 617. 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 1132*

33.4.1 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	CLR_FLAG(1)	0	0	0	0	0	ADC_SEL	0	CHANNEL_VALUE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 736. Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63)

1. This bit implementation is generic based and implemented only for inputs mapped to PIT_RTI event flags.

Table 618. 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
ADC_SEL	This bit selects the ADC number. 0: 10-bit ADC0 is selected 1: 12-bit ADC1 is selected
CHANNEL_ VALUE	These bits provide the ADC channel number to be converted. Valid values are 0b0 to 0b1011111 (decimal 95).

These registers contain the ADC channel number to be converted when the timer event occurs.

When CLR_FLAG is set, CTU ignores any event from PIT_RTI if some triggered conversion is going on in ADC (conversion from CTU to ADC). But if CTU is not busy in sending any trigger to ADC, then it will send the trigger to ADC for conversion of configured channel in CTU_EVTCFGR register. In that case, setting CLR_FLAG bit would not provide flag_ack through software.

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 and automatically disables the DMA connection for the corresponding eMIOS channel.

Note: The CTU tracks issued conversion requests to the ADC. When the ADC is being triggered by the CTU and there is a need to shut down the ADC, the ADC must be allowed to complete conversions before being shut down. This ensures that the CTU is notified of completion; if the ADC is shut down while performing a CTU-triggered conversion, the CTU is not notified and will not be able to trigger further conversions until the device is reset.

33.5 Functional description

This peripheral is used to synchronize ADC conversions with timer events (from eMIOS or PIT_RTI). 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 619*.

Table 619. 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

Table 619. Trigger source (continued)

CTU trigger No.	Module	Source
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
20	eMIOS 0	Channel_20
21	eMIOS 0	Channel_21
22	eMIOS 0	Channel_22
23	PIT_RTI	PIT_3
24	eMIOS 0	Channel_24
25	eMIOS 0	Channel_25
26	eMIOS 0	Channel_26
27	eMIOS 0	Channel_27
28	eMIOS 0	Channel_28
29	eMIOS 0	Channel_29
30	eMIOS 0	Channel_30
31	eMIOS 0	Channel_31
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

Table 619. Trigger source (continued)

CTU trigger No.	Module	Source
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
55	PIT_RTI	PIT_7
56	eMIOS 1	Channel_24
57	eMIOS 1	Channel_25
58	eMIOS 1	Channel_26
59	eMIOS 1	Channel_27
60	eMIOS 1	Channel_28
61	eMIOS 1	Channel_29
62	eMIOS 1	Channel_30
63	eMIOS 1	Channel_31

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_RTI flags are connected. Providing these bits offers the option of clearing PIT_RTI flags by software.

33.5.1 Channel value

The channel value stored in an event configuration register is demultiplexed to 7 bits and then provided to the ADC.

Table 620. CTU-to-ADC Channel Assignment

10-bit ADC			12-bit ADC		
10-bit ADC_0 Signal name	10-bit ADC_0 channel #	Channel # in CTU_EVTCFGRx	12-bit ADC_1 Signal name	12-bit ADC_1 channel #	Channel # in CTU_EVTCFGRx
ADC0_P[0]	CH0	0	ADC1_P[0]	CH0	0
ADC0_P[1]	CH1	1	ADC1_P[1]	CH1	1
ADC0_P[2]	CH2	2	ADC1_P[2]	CH2	2
ADC0_P[3]	CH3	3	ADC1_P[3]	CH3	3
ADC0_P[4]	CH4	4	ADC1_P[4]	CH4	4
ADC0_P[5]	CH5	5	ADC1_P[5]	CH5	5
ADC0_P[6]	CH6	6	ADC1_P[6]	CH6	6
ADC0_P[7]	CH7	7	ADC1_P[7]	CH7	7
ADC0_P[8]	CH8	8	ADC1_P[8]	CH8	8
ADC0_P[9]	CH9	9	ADC1_P[9]	CH9	9
ADC0_P[10]	CH10	10	ADC1_P[10]	CH10	10
ADC0_P[11]	CH11	11	ADC1_P[11]	CH11	11
ADC0_P[12]	CH12	12	ADC1_P[12]	CH12	12
ADC0_P[13]	CH13	13	ADC1_P[13]	CH13	13
ADC0_P[14]	CH14	14	ADC1_P[14]	CH14	14
ADC0_P[15]	CH15	15	ADC1_P[15]	CH15	15
ADC0_S[0]	CH32	32	ADC1_S[0]	CH32	32
ADC0_S[1]	CH33	33	ADC1_S[1]	CH33	33
ADC0_S[2]	CH34	34	ADC1_S[2]	CH34	34
ADC0_S[3]	CH35	35	ADC1_S[3]	CH35	35
ADC0_S[4]	CH36	36	ADC1_S[4]	CH36	36
ADC0_S[5]	CH37	37	ADC1_S[5]	CH37	37
ADC0_S[6]	CH38	38	ADC1_S[6]	CH38	38
ADC0_S[7]	CH39	39	ADC1_S[7]	CH39	39
ADC0_S[8]	CH40	40	ADC1_S[8]	CH40	40
ADC0_S[9]	CH41	41	ADC1_S[9]	CH41	41
ADC0_S[10]	CH42	42	ADC1_S[10]	CH42	42
ADC0_S[11]	CH43	43	ADC1_S[11]	CH43	43
ADC0_S[12]	CH44	44	ADC1_S[12]	CH44	44
ADC0_S[13]	CH45	45			
ADC0_S[14]	CH46	46			
ADC0_S[15]	CH47	47			

Table 620. CTU-to-ADC Channel Assignment (continued)

10-bit ADC			12-bit ADC		
10-bit ADC_0 Signal name	10-bit ADC_0 channel #	Channel # in CTU_EVTCFGRx	12-bit ADC_1 Signal name	12-bit ADC_1 channel #	Channel # in CTU_EVTCFGRx
ADC0_S[16]	CH48	48			
ADC0_S[17]	CH49	49			
ADC0_S[18]	CH50	50			
ADC0_S[19]	CH51	51			
ADC0_S[20]	CH52	52			
ADC0_S[21]	CH53	53			
ADC0_S[22]	CH54	54			
ADC0_S[23]	CH55	55			
ADC0_S[24]	CH56	56			
ADC0_S[25]	CH57	57			
ADC0_S[26]	CH58	58			
ADC0_S[27]	CH59	59			
ADC0_S[28]	CH60	60			
ADC0_S[29]	CH61	61			
ADC0_S[30]	CH62	62			
ADC0_S[31]	CH63	63			
ADC0_X[0]	CH64 : CH71	64:71			
ADC0_X[1]	CH72 : CH79	72:79			
ADC0_X[2]	CH80 : CH87	80:87			
ADC0_X[3]	CH88 : CH95	88:95			

34 Static RAM (SRAM)

34.1 Introduction

This device has up to 256 KB of general-purpose static RAM (SRAM).

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

Except in standby mode, the SRAM is always powered on. In standby mode, the user can decide to retain 8 KB, 40 KB, 64 KB or 96 KB.

34.2 SRAM operating mode

In order to reduce leakage a portion of the SRAM can be switched off/unpowered during standby mode.

34.3 Register memory map

Table 621. Low power configuration

Mode	Configuration
RUN, TEST, SAFE and STOP	The entire SRAM is powered and operational.
STANDBY	In the standby mode, 8 KB, 64 KB, or 96 KB of the SRAM remains powered. This option is software-selectable.
PD	

The L2SRAM occupies 256 KB of memory starting at the base address as shown in *Table 622*.

Table 622. SRAM memory map

Address	Register name	Register description	Size
0x4000_0000 (Base)	—	SRA	up to 256 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).

34.4 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 7-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.

34.4.1 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 623* 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 623. 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	

Note: Above 64 MHz + 4%, additional RAM wait states need to be added. See the MUDCR register description in this reference manual.

34.4.2 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.

34.5 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 34.4: SRAM ECC mechanism*.

34.6 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 34.4: SRAM ECC mechanism*.

35 Flash Memory

35.1 Introduction

The flash memory comprises a platform Flash controller (PFlash) interface and seven flash memory arrays: six arrays of 512 KB for code (CFlash) and one array of 64 KB for data (DFlash). The flash memory architecture of this device is illustrated in *Figure 737*.

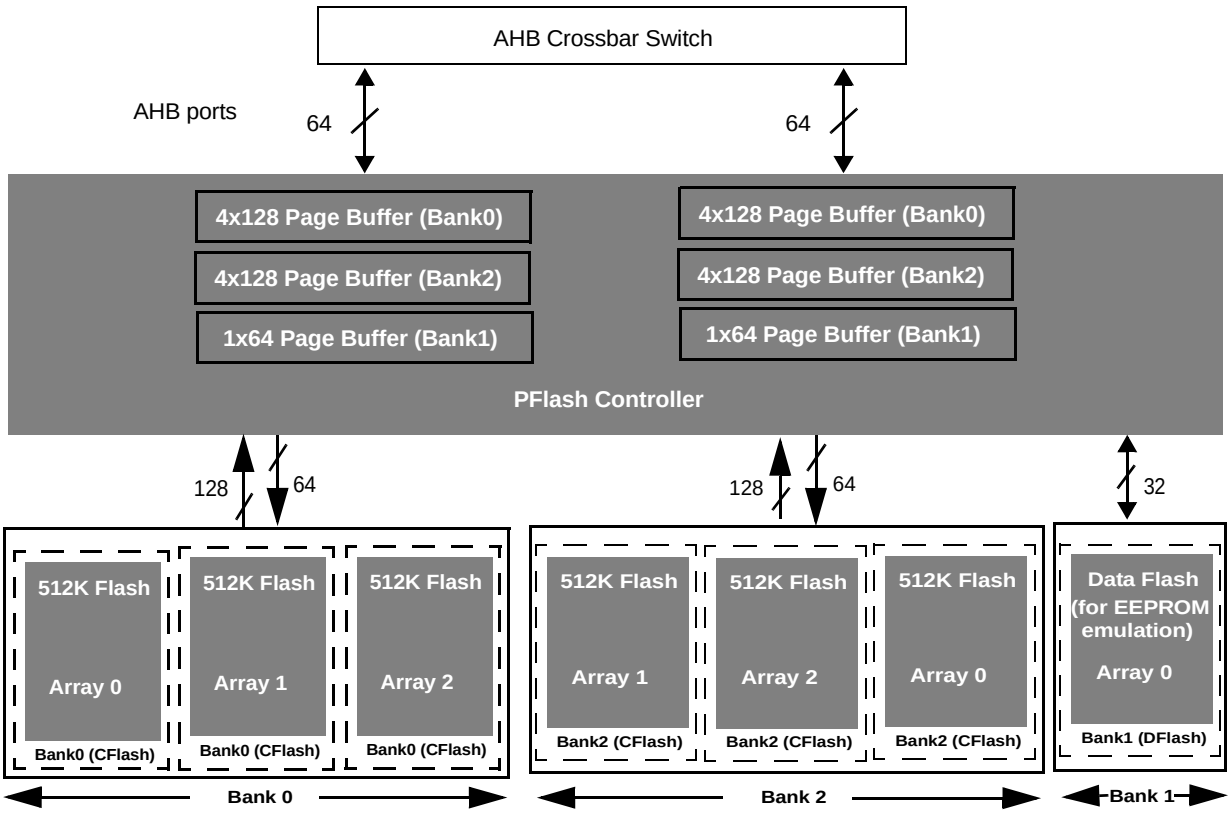


Figure 737. Flash memory architecture

35.2 Code flash memory

35.2.1 Introduction

The primary function of the code flash 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, and 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 and 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 Bus Interface Unit (BIU) and contains the ECC logic and redundancy logic.

A BIU connects the flash memory module to a system bus, and contains all system level customization required for the device application.

35.2.2 Main features

- High Read parallelism (128 bits)
- Error Correction Code (SEC-DED) to enhance Data Retention
- Double Word Program (64 bits)
- Sector erase
- RWW is supported between two code flash memory modules whereas within single bank RWW is not available. Examples of supported RWW below:
  - CF0 (or CF1) and DFlash
  - CF0 and CF1
- Erase Suspend available (Program Suspend not available)
- Software programmable program/erase protection to avoid unwanted writings
- Censored Mode against piracy
- Shadow Sector available
- One-Time Programmable (OTP) area in test flash memory block

35.2.3 Block diagram

The flash memory module contains three Matrix Modules, composed of a three banks: Bank 0, Bank 1, Bank 2, normally used for code storage.

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 3 x128 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.

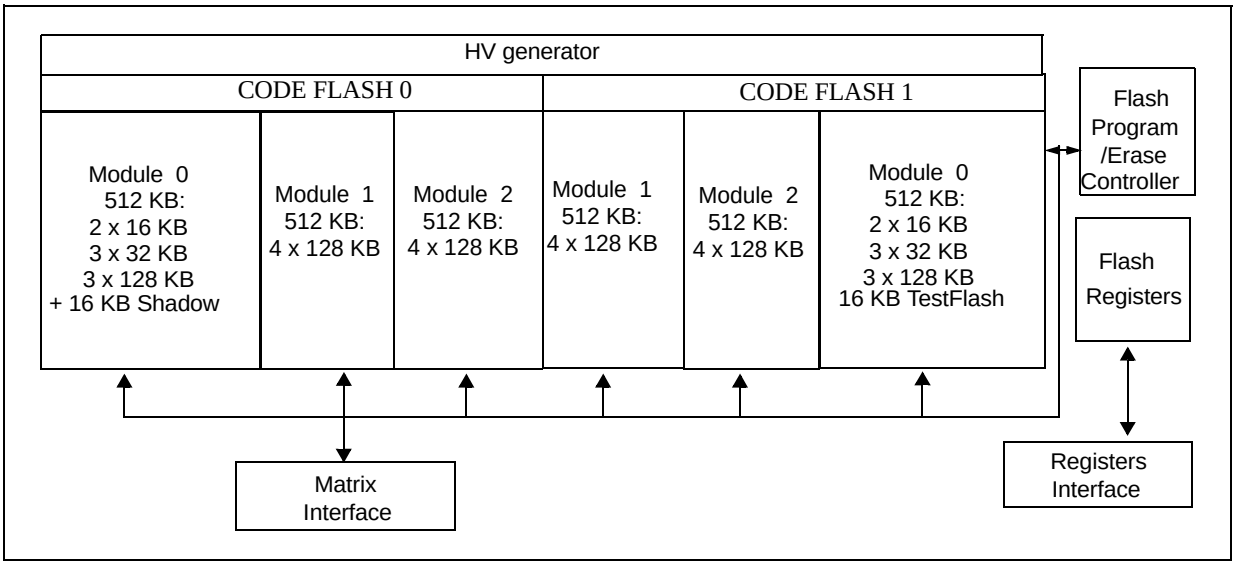


Figure 738. Flash memory module structure

35.2.4 Functional description

35.2.4.1 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 BIU.

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.

35.2.4.2 Flash memory module sectorization

The code flash memory module supports 2 × 1.5 MB of user memory. User can program shadow sector (16 KB). All 16 KB can be programmed by the user.

The flash memory Multi-module is composed of 2 × 1.5 MB Multi-modules, each comprising of the three modules (0, 1 and 2). Read-While-Write is not supported.

The flash memory Multi-Module is divided in 18 sectors including a reserved sector, named TestFlash, in which some One-Time Programmable (OTP) user data are stored in the Code Flash1, as well as a Shadow Sector in which user erasable configuration values can be stored.

The matrix module sectorization is shown in *Table 624*.

Table 624. Flash memory multi module sectorization

Sector	Module	Addresses	Size	Address Space
B0F0	0	0x000000 to 0x007FFF	32 KB	Low Address Space
B0F1	0	0x008000 to 0x00BFFF	16 KB	Low Address Space
B0F2	0	0x00C000 to 0x00FFFF	16 KB	Low Address Space
B0F3	0	0x010000 to 0x017FFF	32 KB	Low Address Space
B0F4	0	0x018000 to 0x01FFFF	32 KB	Low Address Space
B0F5	0	0x020000 to 0x03FFFF	128 KB	Low Address Space
B0F6	0	0x040000 to 0x05FFFF	128 KB	Mid Address Space
B0F7	0	0x060000 to 0x07FFFF	128 KB	Mid Address Space
B0F8	1	0x080000 to 0x09FFFF	128 KB	High Address Space
B0F9	1	0x0A0000 to 0x0BFFFF	128 KB	High Address Space
B0FA	1	0x0C0000 to 0x0DFFFF	128 KB	High Address Space
B0FB	1	0x0E0000 to 0x0FFFFF	128 KB	High Address Space
B0FC	2	0x100000 to 0x11FFFF	128 KB	High Address Space
B0FD	2	0x120000 to 0x13FFFF	128 KB	High Address Space
B0FE	2	0x140000 to 0x15FFFF	128 KB	High Address Space
B0FF	2	0x160000 to 0x17FFFF	128 KB	High Address Space
B2F8	1	0x180000 to 0x19FFFF	128 KB	High Address Space
B2F9	1	0x1A0000 to 0x1BFFFF	128 KB	High Address Space
B2FA	1	0x1C0000 to 0x1DFFFF	128 KB	High Address Space
B2FB	1	0x1E0000 to 0x1FFFFF	128 KB	High Address Space
B2FC	2	0x200000 to 0x21FFFF	128 KB	High Address Space
B2FD	2	0x220000 to 0x23FFFF	128 KB	High Address Space
B2FE	2	0x240000 to 0x25FFFF	128 KB	High Address Space
B2FF	2	0x260000 to 0x27FFFF	128 KB	High Address Space
B2F0	0	0x280000 to 0x287FFF	32 KB	Low Address Space
B2F1	0	0x288000 to 0x28BFFF	16 KB	Low Address Space

Table 624. Flash memory multi module sectorization (continued)

Sector	Module	Addresses	Size	Address Space
B2F2	0	0x28C000 to 0x28FFFF	16 KB	Low Address Space
B2F3	0	0x290000 to 0x297FFF	32 KB	Low Address Space
B2F4	0	0x298000 to 0x29FFFF	32 KB	Low Address Space
B2F5	0	0x2A0000 to 0x2BFFFF	128 KB	Low Address Space
B2F6	0	0x2C0000 to 0x2DFFFF	128 KB	Mid Address Space
B2F7	0	0x2E0000 to 0x2FFFFF	128 KB	Mid Address Space
Reserved	—	0x300000 to 0x3FFFFF	1024 KB	High Address Space
Reserved	—	0x400000 to 0x403FFF	16 KB	High Address Space
Reserved	—	0x404000 to 0x57FFFF	1520 KB	High Address Space
Reserved	—	0x580000 to 0x583FFF	16 KB	High Address Space
Reserved	—	0x584000 to 0x7FFFFF	2544 KB	High Address Space
Reserved	—	0x818000 to 0xBFFFFF	4000 KB	High Address Space
Reserved	—	0x00C000 to 0x00C01FFF	8 KB	High Address Space
Reserved	—	0xC02000 to 0xDFFFFF	2040 KB	High Address Space
Reserved	—	0xE00000 to 0xE03FFF	16 KB	High Address Space
Reserved	—	0xE04000 to 0xE07FFF	16 KB	High Address Space
B2TF	0	0xE08000 to 0xE0BFFF	16 KB	Test Address Space
Reserved	—	0xE0C000 to 0xE0FFFF	16 KB	High Address Space
Reserved	—	0xE10000 to 0xE13FFF	16 KB	High Address Space
Reserved	—	0xE14000 to 0xE17FFF	16 KB	High Address Space
Reserved	—	0xE18000 to 0xE1BFFF	16 KB	High Address Space
Reserved	—	0xE1C000 to 0xE7BFFF	384 KB	High Address Space
Reserved	—	0xE7C000 to 0xE7FFFF	16 KB	High Address Space
Reserved	—	0xE80000 to 0xEFBFFF	496 KB	High Address Space
Reserved	—	0xEFC000 to 0xEFFFFF	16 KB	High Address Space
Reserved	—	0xF00000 to 0xF7BFFF	507 KB	High Address Space
Reserved	—	0xF7C000 to 0xF7FFFF	16 KB	High Address Space
Reserved	—	0xF80000 to 0xFFBFFF	507 KB	High Address Space
B0SH	0	0xFFC000 to 0xFFFFFF	16 KB	Shadow Address Space

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, mid and high address space (as reported in *Table 624*) against program and erase.

35.2.4.2.1 TestFlash block

The TestFlash block exists outside the normal address space and is programmed and read independently of the other blocks. The independent TestFlash block is included 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 625*.

Table 625. TestFlash structure

Name	Description	Address Offset(1)	Size
—	User OTP area	0x000000–0x001FFF	8192 byte
—	Reserved	0x002000–0x003CFF	7424 byte
—	User reserved	0x003D00–0x003DE7	232 byte
NVLML	Nonvolatile Low/Mid address space block Locking register	0x003DE8–0x003DEF	8 byte
NVHBL	Nonvolatile High address space Block Locking register	0x003DF0–0x003DF7	8 byte
NVSLL	Nonvolatile Secondary Low/mid address space block Lock register	0x003DF8–0x003DFF	8 byte
—	User reserved	0x003E00–0x003EFF	256 byte
—	Reserved	0x003F00–0x003FFF	256 byte

1. See device memory map table for base address information of test flash.

Erase of the Test flash memory block is always locked.

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.

35.2.4.2.2 Shadow block

A Shadow block is present in the 544 KB flash memory module.

The Shadow block can be enabled by the BIU.

When the Shadow space is enabled, all the operations are mapped to the Shadow block.

User Mode program and erase of the shadow block are enabled only when MCR.PEAS is high.

The Shadow block may be locked/unlocked against program or erase by using the LML.TSLK and SLL.STSLK registers.

Programming of the Shadow 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 between erases.

Erase of the Shadow block is done similarly to a sector erase.

The Shadow block contains specified data that are needed for user features.

The user area of Shadow block may be used for user defined functions (possibly to store boot code, other configuration words or factory process codes).

The structure of the Shadow sector is detailed in *Table 626*.

Table 626. Shadow sector structure

Name	Description	Address Offset(1)	Size
—	User area	0x0000–0x00F	
—	DCF record Start Fields	0x0010 - 0x001F	
—	NVUSRO_1 Fields	0x0020-0x002F	
—	DCF Record Config Fieldsnvusr_	0x0030-0x00CF	
—	User area	0x00D0-	
—	Reserved	0x003DD0–0x003DD7	8 byte
NVPWD0–1	Nonvolatile Private Censorship PassWord 0–1 registers	0x003DD8–0x003DDF	8 byte
NVSCC0–1	Nonvolatile System Censorship Control 0–1 registers	0x003DE0–0x003DE7	8 byte
—	Reserved	0x003DE8–0x003DFF	24 byte
NVBIU2–3	Nonvolatile Bus Interface Unit 2–3 registers	0x003E00–0x003E0F	16 byte
—	Reserved	0x003E10–0x003E17	8 byte
NVUSRO	Nonvolatile User Options register	0x003E18–0x003E1F	8 byte
—	Reserved	0x003E20–0x003FFF	480 byte

1. See device memory map table for base address information of shadow flash.

35.2.4.3 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 flash memory registers are always available for read, also when the module is in power-down mode (except few documented registers). Most of the flash memory registers are mapped on Flip-Flops and can be read on IPS bus also when the flash memory module is forced in disable mode.

Few flash memory registers (bits MRE, MRV, AIS, EIE and DSI7-0 of UT0, whole UT1 and UT2) are mapped in flash memory SRAM and cannot be read when the flash memory is in disable mode (reading returns indeterminate data)

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 MCR.ERS and MCR.ESUS are high and MCR.PGM is low (Erase Suspend).

Notice that Read-While-Write is not available. 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 Bus Interface Unit.

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 memory Matrix and Read/Write cycles on the registers are possible. On the contrary, registers read/write accesses simultaneous to a flash memory Matrix interlock write are forbidden.

35.2.4.4 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 (MCR.PGM = 1 or MCR.ERS =1) 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 MCR.DONE transitions. MCR.DONE may be polled to determine if the flash memory module has transitioned out of reset. Notice that the registers cannot be written until MCR.DONE is high.

35.2.4.5 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.

Reads from or writes to the module are not possible in power-down mode.

The user may not read some registers (UMISR0–4, UT1–2 and part of UT0) until the power-down mode is exited.

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 set to '1'. The user may resume the erase operation at the time the module is enabled by clearing MCR.ESUS bit. 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.

35.2.4.6 Low power mode

The low power mode turns off most of the DC current sources within the flash memory module.

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.

The user may not read some registers (UMISR0–4, UT1–2 and part of UT0) until the low power mode is exited.

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, bit MCR.ESUS is set to ‘1’. The user may resume the erase operation at the time the module exits low power mode by clearing bit MCR.ESUS. 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.

35.2.5 Register description

The flash memory user registers mapping is shown in *Table 627*.

Table 627. Code flash memory user registers

Address offset(1)	Register	Location
0x0000	Module Configuration Register (MCR)	*on page 1151*
0x0004	Low/Mid address space block Locking reg (LML)	*on page 1157*
0x0008	High address space Block Locking reg (HBL)	*on page 1160*
0x000C	Secondary Low/mid address space block Lock reg (SLL)	*on page 1161*
0x0010	Low/Mid address space block Select reg (LMS)	*on page 1164*
0x0014	High address space Block Select reg (HBS)	*on page 1165*
0x0018	Address Register (ADR)	*on page 1166*
0x001C	Bus Interface Unit reg 0 (BIU0)	*on page 1167*
0x0020	Bus Interface Unit reg 1 (BIU1)	*on page 1167*
0x0024	Bus Interface Unit reg 2 (BIU2)	*on page 1168*
0x0028	Bus Interface Unit reg 3 (BIU3)	*on page 1169*
0x0028–0x0038	Reserved	
0x003C	User Test reg 0 (UT0)	*on page 1170*
0x0040	User Test reg 1 (UT1)	*on page 1172*

Table 627. Code flash memory user registers (continued)

Address offset(1)	Register	Location
0x0044	User Test reg 2 (UT2)	*on page 1172*
0x0048	User Multiple Input Signature Reg 0 (UMISR0)	*on page 1173*
0x004C	User Multiple Input Signature Reg 1 (UMISR1)	*on page 1174*
0x0050	User Multiple Input Signature Reg 2 (UMISR2)	*on page 1174*
0x0054	User Multiple Input Signature Reg 3 (UMISR3)	*on page 1175*
0x0058	User Multiple Input Signature Reg 4 (UMISR4)	*on page 1176*

1. See device memory map table for base address information of code flash0 configuration.

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, BIU, ...), the volatile registers are filled with all ‘1’s and the flash memory initialization ends setting low the PEG bit of MCR.

Caution: Software executing from flash must not write to registers that control flash behavior, e.g., wait state settings or prefetch enable/disable. Doing so can cause data corruption. On SPC564Bxx and SPC56ECxx, these registers include BIU0, BIU1, BIU2.

35.2.5.1 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	EDC (1)	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W	w1c															
Reset	0	0	0	0	0	1	0	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(1)	RWE(1)	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

Figure 739. Module Configuration Register (MCR)

1. This bit is cleared by writing it to a “0”. Writing a “1” to this bit has no effect.

The Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Table 628. 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’. The EDC bit will remain set until cleared by software or a reset occurs. 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, then 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 629*.
LAS	*Low Address Space* The value of the LAS field corresponds to the configuration of the Low Address Space; see *Table 630*.
MAS	*Mid Address Space* The value of the MAS field corresponds to the configuration of the Mid Address Space; see *Table 631*.
EER	*Ecc event ERror* EER provides information on previous reads. If an ECC Double Error detection occurred, the EER bit is set to ‘1’. The EER bit will remain set until cleared by software or a reset occurs. 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, then 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. The RWE bit will remain set until cleared by software or a reset occurs. This bit may not be set to ‘1’ by the user. If RWE is not set, or remains 0, then all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to ‘0’ by writing 1 to the register location. A write of 0 will have no effect. 0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.

Table 628. MCR field descriptions (continued)

Field	Description
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.
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 was properly protected from the Program or Erase operation. If a program operation tries to program a bit which is already at a logic 0 from a logic 1 to a logic 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 successful or Array Integrity Check or Maring Mode completed.

Table 628. MCR field descriptions (continued)

Field	Description
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*) Writing 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 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 628. 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, 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 a high voltage operation. 1: Flash memory is enabled to perform a high voltage operation.

Table 629. Array space size

SIZE[2:0]	Array space size
000	128 KB
001	256 KB
010	512 KB
011	Reserved (1024 KB)
100	1536 KB
101	Reserved (2048 KB)
110	64 KB
111	Reserved

Table 630. Low address space configuration

LAS[2:0]	Low address space sectorization
000	0 KB
001	2 x 128 KB
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	2 x 16 KB + 2 x 32 KB + 2 x 16 KB + 2 x 64 KB

Table 631. Mid address space configuration

MAS	Mid address space sectorization
0	2 x 128 KB or 0 KB
1	Reserved

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash 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 632*.

Table 632. MCR bits set/clear priority levels

Priority level	MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more 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:

1. CPU is stalled when flash is unavailable
2. PEG flag set (stall case) or reset (abort case)
3. 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.

35.2.5.2 Low/Mid address space block Locking register (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 (1)	0	0	0	0	0	0	0	0	0	0	0	X	0	0	X	X

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

Figure 740. Low/Mid address space block Locking register (LML)

1. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xFFFF FFFF. The default value can be reprogrammed by the user.

35.2.5.3 Nonvolatile Low/Mid address space block Locking register (LML)

The Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the SLL register, determine if the block is locked from Program or Erase. An “OR” of LML and SLL determine the final lock status.

The LML register has a related Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for LML. During the reset phase of the flash memory module, the NVLML register content is read and loaded into the LML.

The NVLML register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and eventually used to manage ECC codes.

Table 633. 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 the LME bit is to write a predefined password to the LML register. The predefined password to set the LME bit is 0xA1A11111. Once set, the LME bit will remain set, indicating a status of ENABLED, until a reset operation occurs. 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 (erasing the Test block is forbidden). A value of 1 in the TSLK register signifies that the Test/Shadow block is locked for Program and Erase. A value of 0 in the TSLK register signifies that the Test/Shadow block is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high. 0: Test/Shadow Address Space Block is unlocked and can be modified (also if SLL.STSLK = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.

Table 633. LML field descriptions (continued)

Field	Description
MLK	*Mid address space block LocK 1-0* (Read/Write) 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 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 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 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 544 KB flash memory module 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 SLL.SLK = 0). 1: Low Address Space Block is locked and cannot be modified.

35.2.5.4 High address space Block Locking register (HBL)

Offset: 0x0008 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset(1)	0	0	0	0	0	0	0	0	0	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	HLK11	HLK10	HLK9	HLK8	HLK7	HLK6	HLK5	HLK4	HLK3	HLK2	HLK1	HLK0
W																
Reset	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X

Figure 741. High address space Block Locking register (HBL)

1. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xFFFF FFFF. The default value can be reprogrammed by the user.

35.2.5.5 Nonvolatile High address space Block Locking register (NVHBL)

The High Address Space Block Locking register provides a means to protect blocks from being modified.
The HBL register has a related Nonvolatile High Address Space Block Locking register located in TestFlash that contains the default reset value for HBL. During the reset phase of the flash memory module, the NVHBL register content is read and loaded into the HBL.

The NVHBL register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and eventually used to manage ECC codes.

Table 634. HBL field descriptions

Field	Description
HBE	*High address space Block Enable* (Read Only) This bit is used to enable the Lock registers (HLK5-0) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the HBE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B22222 must be written to the HBL register. 0: High Address Locks are disabled: HLK5-0 cannot be written. 1: High Address Locks are enabled: HLK5-0 can be written.
HLK11-0	**HLK11-0**: *High address space block lock 11-0* (Read/Write) These bits are used to lock the blocks of High Address Space from Program and Erase. HLK11-8 are not used for this memory cut. A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive Program and Erase pulses. The HLK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the HLK registers. The HLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the HLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the HLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the 544 KB flash memory module bits HLK11-8 are read-only and locked at 1. HLK is not writable unless HBE is high. 0: High Address Space Block is unlocked and can be modified. 1: High Address Space Block is locked and cannot be modified.

35.2.5.6 Secondary Low/mid address space block Locking register (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	SMK1	SMK0
W																
Reset (1)	0	0	0	0	0	0	0	0	0	0	0	X	0	0	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
W																
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 742. Secondary Low/mid address space block Locking register (SLL)

1. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xFFFF FFFF. The default value can be reprogrammed by the user.

35.2.5.7 Nonvolatile Secondary Low/mid address space block Locking register (SLL)

The Secondary Low/Mid Address Space Block Locking register provides an alternative means to protect blocks from being modified. These bits, along with bits in the LML register, determine if the block is locked from Program or Erase. An “OR” of LML and SLL determine the final lock status.

The 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 NVSLL register content is read and loaded into the SLL.

The NVSLL register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and eventually used to manage ECC codes.

Table 635. SLL field descriptions

Field	Description
SLE	*Secondary Low/mid address space block Enable* (Read Only) 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 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* (Read/Write) This bit is used as an alternate means to lock the block of Test and Shadow Address Space from Program and Erase (erasing the Test block is forbidden). A value of 1 in the STSLK register signifies that the Test/Shadow block is locked for Program and Erase. A value of 0 in the STSLK register signifies that the Test/Shadow block is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high. 0: Test/Shadow Address Space Block is unlocked and can be modified (also if LML.TSLK = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.

Table 635. SLL field descriptions (continued)

Field	Description
SMK[1:0]	*Secondary Mid address space block locK 1-0* (Read/Write) 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. When an SMK bit is 0, it signifies that the corresponding block is available to receive program and erase pulses. The SMK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes 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 LML.MLK = 0). 1: Mid Address Space Block is locked and cannot be modified.
SLK[15:0]	*Secondary Low address space block locK 15-0* (Read/Write) 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 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 544 KB flash memory module 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 LML.LLK = 0). 1: Low Address Space Block is locked and cannot be modified.

35.2.5.8 Low/Mid address space block Select register (LMS)

Offset: 0x0010 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	MSL1	MSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 743. Low/Mid address space block Select register (LMS)

The Low/Mid Address Space Block Select register provides a means to select blocks to be operated on during erase.

Table 636. LMS field descriptions

Field	Description
MSL[1:0]	*Mid address space block SeLect 1-0* (Read/Write) 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[15:0]	*Low address space block SeLect 15-0* (Read/Write) 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. In the 544 KB flash memory module 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.

35.2.5.9 High address space Block Select register (HBS)

Offset: 0x0014 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	HSL11	HSL10	HSL9	HSL8	HSL7	HSL6	HSL5	HSL4	HSL3	HSL2	HSL1	HSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 744. High address space Block Select register (HBS)

The High Address Space Block Select register provides a means to select blocks to be operated on during erase.

Table 637. HBS field descriptions

Field	Description
HSL11-0	*High address space block SeLect 11-0* (Read/Write) 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. HSL11-8 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 HSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. In the 544 KB Flash module bits HSL11-8 are read-only and locked at 0. 0: High Address Space Block is unselected for Erase. 1: High Address Space Block is selected for Erase.

35.2.5.10 Address Register (ADR)

Offset: 0x0018 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

Figure 745. Address Register (ADR)

The Address Register 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.

Table 638. ADR field descriptions

Field	Description
AD[22:3]	*ADdress 22-3* (Read Only) The Address Register provides the first failing address in the event of ECC error (MCR.EER set) or the first failing address in the event of RWW error (MCR.RWE set), or the address of a failure that may have occurred in a FPEC operation (MCR.PEG cleared). The Address Register also provides the first address at which an ECC single error correction occurs (MCR.EDC set). The ECC double error detection takes the highest priority, followed by the RWW error, the FPEC error and the ECC single error correction. When accessed ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in the *Table 639*. 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.
Bits 29:31	*Reserved* (Read Only). Writing these bits has no effect and read these bits always outputs 0.

Table 639. ADR content: priority list

Priority level	Error flag	ADR content
1	MCR.EER = 1	Address of first ECC Double Error
2	MCR.RWE = 1	Address of first RWW Error
3	MCR.PEG = 0	Address of first FPEC Error
4	MCR.EDC = 1	Address of first ECC Single Error Correction

35.2.5.11 Bus Interface Unit 0 register (BIU0)

Offset: 0x001C Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BI031	BI030	BI029	BI028	BI027	BI026	BI025	BI024	BI023	BI022	BI021	BI020	BI019	BI018	BI017	BI016
Reset (1)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI015	BI014	BI013	BI012	BI011	BI010	BI009	BI008	BI007	BI006	BI005	BI004	BI003	BI002	BI001	BI000
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 746. Bus Interface Unit 0 register (BIU0)

1. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xXXXX XXXX. The default value can be reprogrammed by the user.

The Bus Interface Unit 0 Register provides a means for BIU specific information or BIU configuration information to be stored. Please refer to *Section 35.4.3.1*, for more information about register description.

Table 640. BIU0 field descriptions

Field	Description
BI0[31:00]	*Bus Interface unit 0 31-00* (Read/Write) The writability of the bits in this register can be locked.

35.2.5.12 Bus Interface Unit 1 register (BIU1)

Offset: 0x0020 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BI131	BI130	BI129	BI128	BI127	BI126	BI125	BI124	BI123	BI122	BI121	BI120	BI119	BI118	BI117	BI116
Reset (1)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI115	BI114	BI113	BI112	BI111	BI110	BI109	BI108	BI107	BI106	BI105	BI104	BI103	BI102	BI101	BI100
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 747. Bus Interface Unit 1 register (BIU1)

1. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xXXXX XXXX. The default value can be reprogrammed by the user.

The Bus Interface Unit 1 Register provides a means for BIU specific information or BIU configuration information to be stored. Please refer to *Section 35.4.3.2*, for more information about register description.

Table 641. BIU1 field descriptions

Field	Description
BI1[31:00]	*Bus Interface unit 1 31-00* (Read/Write) The writability of the bits in this register can be locked.

35.2.5.13 Bus Interface Unit 2 register (BIU2)

Offset: 0x0024(1) Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R(2) W	BI231	BI230	BI229	BI228	BI227	BI226	BI225	BI224	BI223	BI222	BI221	BI220	BI219	BI218	BI217	BI216
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI215	BI214	BI213	BI212	BI211	BI210	BI209	BI208	BI207	BI206	BI205	BI204	BI203	BI202	BI201	BI200
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 748. Bus Interface Unit 2 register (BIU2)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xXXXX XXXX. The default value can be reprogrammed by the user.

The Bus Interface Unit 2 Register provides a means for BIU specific information or BIU configuration information to be stored. Please refer to *Section 35.4.3.3*, for more information about register description.

The BIU2 register has a related Nonvolatile Bus Interface Unit 2 register located in the Shadow Sector that contains the default reset value for BIU2. During the reset phase of the flash memory module, the NVBIU2 register content is read and loaded into the BIU2.

The NVBIU2 register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and eventually used to manage ECC codes.

Table 642. BIU2 field descriptions

Field	Description
BI2[31:00]	*Bus Interface unit 2 31-00* (Read/Write) The BI2[31:00] generic registers are reset based on the information stored in NVBIU2. The writability of the bits in this register can be locked.

35.2.5.14 Bus Interface Unit 3 register (BIU3)

Offset: 0x0028(1) Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BI331	BI330	BI329	BI328	BI327	BI326	BI325	BI324	BI323	BI322	BI321	BI320	BI319	BI318	BI317	BI316
Reset(2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BI315	BI314	BI313	BI312	BI311	BI310	BI309	BI308	BI307	BI306	BI305	BI304	BI303	BI302	BI301	BI300
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 749. Bus Interface Unit 0 register (BIU0)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xFFFF FFFF. The default value can be reprogrammed by the user.

The Bus Interface Unit 3 Register provides a means for BIU specific information or BIU configuration information to be stored.

The BIU3 register has a related Nonvolatile Bus Interface Unit 3 register located in the Shadow Sector that contains the default reset value for BIU3. The NVBIU3 register is read during the reset phase of the flash memory module and loaded into the BIU3.

The NVBIU3 register is a 64-bit register, the 32 most significative bits of which (bits 63:32) are ‘don’t care’ and eventually used to manage ECC codes.

Table 643. BIU3 field descriptions

Field	Description
BI331-00	*Bus Interface unit 3 31-00* (Read/Write) The BI331-00 generic registers are reset based on the information stored in NVBIU3. The writability of the bits in this register can be locked. The use of this bus is SoC specific.

35.2.5.15 User Test 0 register (UT0)

Offset: 0x003C 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	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W																
Reset	0	0	0	0	0	0	0	0	0	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

Figure 750. User Test 0 register (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 control of 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 MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 644. UT0 field descriptions

Field	Description
UTE	*User Test Enable* (Read/Clear) This status bit gives indication when User Test is enabled. All bits in UT0-2 and UMISR0-4 are locked when this bit is 0. This bit is not writeable to a 1, but may be cleared. The reset value is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and remains enabled until it is cleared by a register write. For UTE the password 0xF9F99999 must be written to the UT0 register.
Bit 1:7	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
DSI[7:0]	*Data Syndrome Input 7-0* (Read/Write) 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 MCR.DONE or 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.
Bits 16:24	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
Bit 25	*Reserved* (Read/Write). This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 644. UT0 field descriptions (continued)

Field	Description
MRE	*Margin Read Enable* (Read/Write) 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 MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect. 0: Margin reads are not enabled 1: Margin reads are enabled.
MRV	*Margin Read Value* (Read/Write) If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever MCR.DONE or UT0.AID are low: 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* (Read/Write) EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever MCR.DONE or 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* (Read/Write) 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 MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect. 0: Array Integrity sequence is proprietary sequence. 1: Array Integrity is sequential.
AIE	*Array Integrity Enable* (Read/Write) AIE set to '1' starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if MCR.ERS, MCR.PGM and MCR.EHV are all low. 0: Array Integrity Checks, 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* (Read Only) 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 (UMISR0-4) can be checked. 0: Array Integrity Check is on-going. 1: Array Integrity Check is done.

35.2.5.16 User Test 1 register (UT1)

Offset: 0x0040 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
Reset	0	0	0	0	0	0	0	0	0	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	DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 751. User Test 1 register (UT1)

The User Test 1 Register allows to enable the checks on the ECC logic related to the 32 LSB of the Double Word.

The User Test 1 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 645. UT1 field descriptions

Field	Description
DAI[31:00]	*Data Array Input 31-0* (Read/Write) 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.

35.2.5.17 User Test 2 register (UT2)

Offset: 0x0044 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI63	DAI62	DAI61	DAI60	DAI59	DAI58	DAI57	DAI56	DAI55	DAI54	DAI53	DAI52	DAI51	DAI50	DAI49	DAI48
Reset	0	0	0	0	0	0	0	0	0	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	DAI47	DAI46	DAI45	DAI44	DAI43	DAI42	DAI41	DAI40	DAI39	DAI38	DAI37	DAI36	DAI35	DAI34	DAI33	DAI32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 752. User Test 2 register (UT2)

The User Test 2 Register allows to enable the checks on the ECC logic related to the 32 MSB of the Double Word.

The User Test 2 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 646. UT2 field descriptions

Field	Description
DAI[63:32]	*Data Array Input 63-32* (Read/Write) 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 to 0. 1: The array bit is forced to 1.

35.2.5.18 User Multiple Input Signature Register 0 (UMISR0)

Offset: 0x0048 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS31	MS30	MS29	MS28	MS27	MS26	MS25	MS24	MS23	MS22	MS21	MS20	MS19	MS18	MS17	MS16
Reset	0	0	0	0	0	0	0	0	0	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	MS15	MS14	MS13	MS12	MS11	MS10	MS09	MS08	MS07	MS06	MS05	MS04	MS03	MS02	MS01	MS00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 753. User Multiple Input Signature Register 0 (UMISR0)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity. The User Multiple Input Signature Register 0 represent the bits 31-0 of the word. The UMISR0 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 647. UMSIR0 field descriptions

Field	Description
MS31:00	*Multiple input Signature 31-00* (Read/Write) 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 UMISR0 register.

35.2.5.19 User Multiple Input Signature Register 1 (UMISR1)

Offset: 0x004C Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
Reset	0	0	0	0	0	0	0	0	0	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	MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 754. User Multiple Input Signature Register 1 (UMISR1)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity. The User Multiple Input Signature Register 1 represent the ECC bits of the 32 bits word: bits 6-0 are the ECC bits for the Word; bits 10 and 11 of MISR are respectively the double and single ECC error detection.

The UMISR1 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 648. UMISR1 field descriptions

Field	Description
MS063:032	*Multiple input Signature 063-032* (Read/Write) These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the flash memory. The MS can be seeded to any value by writing the UMISR1 register.

35.2.5.20 User Multiple Input Signature Register 2 (UMISR2)

Offset: 0x0050 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS095	MS094	MS093	MS092	MS091	MS090	MS089	MS088	MS087	MS086	MS085	MS084	MS083	MS082	MS081	MS080
Reset	0	0	0	0	0	0	0	0	0	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	MS079	MS078	MS077	MS076	MS075	MS074	MS073	MS072	MS071	MS070	MS069	MS068	MS067	MS066	MS065	MS064
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 755. User Multiple Input Signature Register 2 (UMISR2)

The Multiple Input Signature Register provides a means to evaluate the Array Integrity.

The User Multiple Input Signature Register 2 represents the bits 95:64 of the whole 144 bits word (2 Double Words including ECC).

The UMISR2 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 649. UMISR2 field descriptions

Field	Description
MS095:064	*Multiple input Signature 095-064* (Read/Write) 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 UMISR2 register.

35.2.5.21 User Multiple Input Signature Register 3 (UMISR3)

Offset: 0x0054 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS127	MS126	MS125	MS124	MS123	MS122	MS121	MS120	MS119	MS118	MS117	MS116	MS115	MS114	MS113	MS112
Reset	0	0	0	0	0	0	0	0	0	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	MS111	MS110	MS109	MS108	MS107	MS106	MS105	MS104	MS103	MS102	MS101	MS100	MS099	MS098	MS097	MS096
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 756. User Multiple Input Signature Register 3 (UMISR3)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity.

The User Multiple Input Signature Register 3 represents the bits 127:96 of the whole 144 bits word (2 Double Words including ECC).

The UMISR3 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 650. UMISR3 field descriptions

Field	Description
MS127:096	*Multiple input Signature 127-096* (Read/Write) 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 UMISR3 register.

35.2.5.22 User Multiple Input Signature Register 4 (UMISR4)

Offset: 0x0058 Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS159	MS158	MS157	MS156	MS155	MS154	MS153	MS152	MS151	MS150	MS149	MS148	MS147	MS146	MS145	MS144
Reset	0	0	0	0	0	0	0	0	0	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	MS143	MS142	MS141	MS140	MS139	MS138	MS137	MS136	MS135	MS134	MS133	MS132	MS131	MS130	MS129	MS128
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 757. User Multiple Input Signature Register 4(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 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 UMISR4 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 651. UMISR4 field descriptions

Field	Description
f	*Multiple input Signature 159-128* (Read/Write) 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 UMISR4 register.

35.2.5.23 Nonvolatile private censorship PassWord 0 register (NVPWD0)

Offset: 0x03DD8[1] Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD31	PWD30	PWD29	PWD28	PWD27	PWD26	PWD25	PWD24	PWD23	PWD22	PWD21	PWD20	PWD19	PWD18	PWD17	PWD16
Reset (2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWD15	PWD14	PWD13	PWD12	PWD11	PWD10	PWD09	PWD08	PWD07	PWD06	PWD05	PWD04	PWD03	PWD02	PWD01	PWD00
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 758. Nonvolatile private censorship PassWord 0 register (NVPWD0)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xFEED_FACE. The default value can be reprogrammed by the user.

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.

Table 652. NVPWD0 field descriptions

Field	Description
PWD[31:00]	*PassWorD 31-00* (Read/Write) The PWD31-00 registers represent the 32 LSB of the Private Censorship Password.

35.2.5.24 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.

Note: In a secured device, starting with a serial boot, it is possible to read the content of the four flash memory 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 memory address cannot be accessed.

Offset: 0x03DDC(1) Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD63	PWD62	PWD61	PWD60	PWD59	PWD58	PWD57	PWD56	PWD55	PWD54	PWD53	PWD52	PWD51	PWD50	PWD49	PWD48
Reset (2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWD47	PWD46	PWd45	PWD44	PWD43	PWD42	PWD41	PWD40	PWD39	PWD38	PWD37	PWD36	PWD35	PWD34	PWD33	PWD32
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 759. Nonvolatile private censorship PassWord 1 register (NVPWD1)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xCAFE_BEEF. The default value can be reprogrammed by the user.

Table 653. NVPWD1 field descriptions

Field	Description
PWD[63:32]	*PassWorD 63-32* (Read/Write) The PWD63-32 registers represent the 32 MSB of the Private Censorship Password.

35.2.5.25 Nonvolatile System Censoring Information 0 register (NVSCC0)

Offset: 0x03DE0(1) Delivery value: 0x55AA_55AA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC15	SC14	SC13	SC12	SC11	SC10	SC9	SC8	SC7	SC6	SC5	SC4	SC3	SC2	SC1	SC0
Reset (2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CW15	CW14	CW13	CW12	CW11	CW10	CW9	CW8	CW7	CW6	CW5	CW4	CW3	CW2	CW1	CW0
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 760. Nonvolatile System Censoring Information 0 register (NVSCC0)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0x55AA_55AA. The default value can be reprogrammed by the user.

The nonvolatile system censoring information 0 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.

Table 654. NVSCC0 field descriptions

Field	Description
SC[15:0]	*Serial Censorship control word 15-0* (Read/Write) 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 15-0* (Read/Write) 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.

35.2.5.26 Nonvolatile System Censoring Information 1 register (NVSCC1)

Offset: 0x03DE4(1) Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC31	SC30	SC29	SC28	SC27	SC26	SC25	SC24	SC23	SC22	SC21	SC20	SC19	SC18	SC17	SC16
Reset (2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CW31	CW30	CW29	CW28	CW27	CW26	CW25	CW24	CW23	CW22	CW21	CW20	CW19	CW18	CW17	CW16
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 761. Nonvolatile System Censoring Information 1 register (NVSCC1)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0x55AA_55AA. The default value can be reprogrammed by the user.

The nonvolatile System Censoring Information 1 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.

Table 655. NVSCC1 field descriptions

Field	Description
SC[31:16]	*Serial Censorship control word 31-16* (Read/Write) 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 31-16* (Read/Write) 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.

35.2.5.27 Nonvolatile User Options register (NVUSRO)

Offset: 0x03E18(1) Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WATCHDOG_EN	UO30	PAD_3V5V[0]	PAD_3V5V[1]	UO27	UO26	UO25	UO24	UO23	UO22	UO21	UO20	UO19	UO18	UO17	UO16
Reset (2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	UO15	UO14	UO13	UO12	UO11	STCU_EN	UO09	UO08	UO07	UO06	UO05	UO04	UO03	UO02	UO01	UO01
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 762. Nonvolatile User Options register (NVUSRO)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xXXXX_XXXX. The default value can be reprogrammed by the user.

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 eventually used to manage ECC codes.

Table 656. NVUSRO field descriptions

Field	Description
UO	*User Options* (Read/Write) The UO generic field is reset based on the information stored in NVUSRO.
STCU_EN	Self MBIST Enable (Read/Write) 1: Disable after reset 0: Enable after reset Default manufacturing value before flash memory initialization is ‘1’.

Table 656. NVUSRO field descriptions (continued)

Field	Description
UO[20:04]	*User Options 20:04* (Read/Write) The UO20-4 generic registers are reset based on the information stored in NVUSRO.
PAD3V5V[0]	High voltage supply for $V_{DD_HV_A}$ domain 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. This bit should be set to actual supply voltage for this domain. Mismatch in bit value and supply voltage can change I/O drive strength, but no damage should happen.
PAD3V5V[1]	High voltage supply for $V_{DD_HV_B}$ domain 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. This bit should be set to actual supply voltage for this domain. Mismatch in bit value and supply voltage can change I/O drive strength, but no damage should happen.
UO[01]	*User Options 01* (Read/Write) The UO01 generic register is reset based on the information stored in NVUSRO.
WATCHDOG_EN	0: Disable after reset 1: Enable after reset Default manufacturing value before flash memory initialization is ‘1’

35.2.5.28 Nonvolatile User Options register 1(NVUSRO_1)

Offset: 0x0020(1) Access: Read / Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	UO31	UO30	UO29	UO28	UO27	UO26	UO25	UO24	UO23	UO22	UO21	UO20	UO19	UO18	UO17	UO16
Reset (2)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	UO15	UO14	UO13	UO12	UO11	UO10	UO09	UO08	UO07	UO06	UO05	UO04	UO03	UO02	UO01	CSE_RUN_MODE
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 763. Nonvolatile User Options register 1(NVUSRO_1)

1. See device memory map table for base address information of shadow flash.
2. Reset values labeled “X” are loaded from the NVLML value which is preprogrammed at the factory. The default value from the factory for this register is 0xFFFF_FFFF. The default value can be reprogrammed by the user.

The Nonvolatile User Options Register 1 contains configuration information for the user application.

Table 657. NVUSRO_1 field descriptions

Field	Description
CSE_RUN_MODE	1: CSE runs in parallel mode 0: CSE runs in serial mode Default manufacturing value before flash memory initialization is '1'.
UO[31:1]	*User Options 31-1* (Read/Write) The UO31-1 generic registers are reset based on the information stored in NVUSRO_1.

35.2.5.29 Programming NVUSRO_1 and STCU fault grading parameters

The following shadow addresses need to be programmed to configure the NVUSRO_1 register and STCU fault grading parameters.

Table 658. Programming NVUSRO_1 and STCU fault grading parameters

Address offset(1) (Byte)	flash memory content(2) (word)	Comment
0x0010	0x05AA_55AF	START word
0x0014	0x0000_0000	
0x0018	0xFFFFFFFF	
0x001C	0xFFFFFFFF	
0x0020	NVUSRO_1	NVUSRO_1
0x0024	0x00400000	
0x0028		
0x002C		
0x0030	0x7f0e0000	UPDATE STCU MBIST 0 CTR
0x0034	0x00080300	
0x0038	0xFFFFFFFF	
0x003C	0xFFFFFFFF	
0x0040	0x10000000	UPDATE STCU CFG Register
0x0044	0x0008000c	
0x0048	0xFFFFFFFF	
0x004C	0xFFFFFFFF	
0x0050	0x825a132b	UPDATE STCU MBSEK Register
0x0054	0x0008004c	
0x0058	0xFFFFFFFF	
0x005C	0xFFFFFFFF	

Table 658. Programming NVUSRO_1 and STCU fault grading parameters

Address offset(1) (Byte)	flash memory content(2) (word)	Comment
0x0060	0xfffffffe	UPDATE STCU MBSL Register
0x0064	0x0008003c	
0x0068	0xFFFFFFFF	
0x006C	0xFFFFFFFF	
0x0070	0x000000ff	Update STCU MBSH Register
0x0074	0x00080040	
0x0078	0xFFFFFFFF	
0x007C	0xFFFFFFFF	
0x0080	0xfffffffe	UPDATE STCU MBEL Register
0x0084	0x00080044	
0x0088	0xFFFFFFFF	
0x008C	0xFFFFFFFF	
0x0090	0x000000ff	Update STCU MBEH Register
0x0094	0x00080048	
0x0098	0xFFFFFFFF	
0x009C	0xFFFFFFFF	
0x00A0	0x00000001	Update STCU WDG Register
0x00A4	0x00080010	
0x00A8	0xFFFFFFFF	
0x00AC	0xFFFFFFFF	
0x00B0	0x1a60d097	Update STCU CRC Register
0x00B4	0x00080014	
0x00B8	0xFFFFFFFF	
0x00BC	0xFFFFFFFF	
0x00C0	0xFFFFFFFF	Finish WORD
0x00C4	0xFFFFFFFF	
0x00C8	0xFFFFFFFF	
0x00CC	0xFFFFFFFF	

1. See device memory map table for base address information of code flash0 shadow sector.
2. Default settings are erased (0xFFFF_FFFF)

35.2.6 STCU programming using Flash

The Shadow Flash contains the data for STCU programming. *Table 659* shows the interpretation of the Flash read bus by the STCU. This results in the memory usage show in *Table 660*. Each write command for the STCU consists of 32 bits for write data, and 32 bits

for write address and chip select fields. The next 64 bits are reserved (this is to avoid data multiplexing). One bit in the data structure is the STOP flag. The STCU will process all commands until it detects a STOP flag. The command with the stop flag will not cause a write operation on the DCF bus.

So an empty Shadow Flash will not create a series of commands, instead it will just cause the STCU to stop scanning the Flash.

It has to detect the start word 0x05AA55AF at the location offset + 0x0 for execution of write command.

Table 659. Interpretation of Flash read data

127:64	63:49	48:34	33	32	31:0
RSVD	CS[14:0]	ADDR[16:2]	RSVD	STOP	WDATA[31:0]

Table 660. Storing STCU data in shadow flash

ADDR Offset	Data			
16n + 0xC				
16n + 0x8				
16n + 0x4	Reserved			1
16n + 0x0	Reserved			
16n-1 + 0xC				
16n-1 + 0x8				
16n-1 + 0x4	CS[14:0]	ADDR[16:2]	RSVD	0
16n-1 + 0x0	WDATA[31:0]			
...	.....			
0x2C				
0x28				
0x24	CS[14:0]	ADDR[16:2]	RSVD	STOP
0x20	WDATA[31:0]			
0x1C				
0x18				
0x14	CS[14:0]	ADDR[16:2]	RSVD	STOP
0x10	WDATA[31:0]			
0xC				
0x8				
0x4	0x00000000			STOP=0
0x0	0x05AA55AF			

35.2.7 Programming considerations

35.2.7.1 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 RWE of the MCR 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 module 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 EHV in MCR or AIE in UT0.

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 661*.

Table 661. 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, HBS	MCR.EHV
Array integrity check	None	LMS, HBS	UT0.AIE
Margin read	UT0.MRE	UT0.MRV + LMS, HBS	UT0.AIE
ECC Logic Check	UT0.EIE	UT0.DSI, UT1, UT2	UT0.AIE

Once bit MCR.EHV (or UT0.AIE) is set, all the operands can no more be modified until bit MCR.DONE (or UT0.AID) is high.

In general each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for bit MCR.DONE (or UT0.AID) to go high.
2. Check operation result: check bit MCR.PEG (or compare UMISR0-4 with expected value).
3. Switch off FPEC by resetting MCR.EHV (or UT0.AIE).
4. Deselect current operation by clearing MCR.PGM/ERS (or UT0.MRE/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.

35.2.7.1.1 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 MCR.PEG = 1.

7. Write a logic 0 to the MCR.EHV bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR.PGM bit to terminate the program operation.

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 MCR.PEAS to be set/cleared.

An interlock write must be performed before setting MCR.EHV. The user may terminate a program sequence by clearing MCR.PGM prior to setting MCR.EHV.

After the interlock write, additional writes only affect the data to be programmed at the word location determined by address bit 2. Unwritten locations default to a data value of 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 1 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 */
```

35.2.7.1.2 Sector erase

Erase changes the value stored in all bits of the selected block(s) to logic 1.

An erase sequence operates on any combination of blocks (sectors) in the low, mid or high address space, or the shadow block (if available). The test block cannot be erased.

The erase sequence is fully automated within the flash 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 register(s) in LMS or HBS registers.
   If the shadow block is to be erased, this step may be skipped, and LMS and HBS are 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 2 Erase of sectors B0F1 and B0F2

```
MCR   = 0x00000004;/* Set ERS in MCR: Select Operation */
LMS   = 0x00000006;/* Set LSL2-1 in LMS: Select Sectors to erase */
(0x000000)= 0xFFFFFFFF;/* Latch a flash 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 3 Sector erase suspend

```
MCR  = 0x00000007;/* Set ESUS in MCR: Erase Suspend */
do              /* Loop to wait for DONE=1 */
{ tmp= MCR;     /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Notice that there is no need to clear MCR.EHV and MCR.ERS in order to perform reads during erase suspend.

The erase sequence is resumed by writing a logic 0 to MCR.ESUS.

MCR.EHV must be set to ‘1’ before MCR.ESUS can be cleared to resume the operation.

The module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

Example 4 Sector erase resume

```
MCR  = 0x00000005;/* Reset ESUS in MCR: Erase Resume */
```

35.2.7.1.3 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 (RWE of MCR set).

It is not allowed to perform User Test operations on the Test and Shadow blocks.

35.2.7.1.3.1 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 UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing ‘1’s to the appropriate register(s) in LMS or HBS registers.
   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, HBS) and Lock (LML, SLL, HBL) 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 5 Array integrity check of sectors B0F1 and B0F2

```
UT0   = 0xF9F99999;/* Set UTE in UT0: Enable User Test */
LMS   = 0x00000006;/* Set LSL2-1 in LMS: Select Sectors */
UT0   = 0x80000002;/* Set AIE in UT0: Operation Start */
do            /* Loop to wait for AID=1 */
{ tmp= UT0;  /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0= UMISR0;/* Read UMISR0 content*/
data1= UMISR1;/* Read UMISR1 content*/
data2= UMISR2;/* Read UMISR2 content*/
```

```
data3 = UMISR3;/* Read UMISR3 content*/
data4 = UMISR4;/* Read UMISR4 content*/
UT0   = 0x00000000;/* Reset UTE and AIE in UT0: Operation End */
```

35.2.7.1.3.2 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 module 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 UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1's to the appropriate register(s) in LMS or HBS registers.

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, HBS) and Lock (LML, SLL, HBL) 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 6 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.

35.2.7.1.3.3 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 UTE in UT0 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 7 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 */
```

35.2.7.2 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.

35.2.7.2.1 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.

35.2.7.3 EEprom emulation

35.2.7.4 Eprom 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 chosen 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.

The following table shows a set of Double Words sharing the same ECC value:

Table 662. Bits 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

Table 662. Bits Manipulation: Double Words with the same ECC value

Double Word	ECC All ‘1’s No Error
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 recommended for safety reasons to reserve at least 3 sectors to this purpose.

35.2.7.4.1 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.

35.2.7.5 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.

35.2.7.5.1 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 (Low/Mid Address Space Block Lock Register) or HBL (High Address Space Block Lock Register) registers.

An alternate means to enable software locking for blocks of Low Address Space only is through the SLL (Secondary Low/Mid Address Space Block Lock Register).

All these registers have a nonvolatile image stored in TestFlash (NVLML, NVHBL, 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.

35.2.7.5.2 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 eventually 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.

35.3 Data flash memory

35.3.1 Introduction

The primary function of the Data flash 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, and a means of selectively sensing (reading) the charge stored in these elements.

The Data flash memory module is arranged as two functional units: the flash memory core and 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 flash memory core is organized including ECC correction code. 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.

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 Bus Interface Unit (BIU) and contains the ECC logic and redundancy logic.

A BIU connects the flash memory module to a system bus, and contains all system level customization required for the device application. The flash memory module is generic and requires a BIU to configure it for different device applications. A BIU is not included as a part of the flash memory module.

35.3.2 Main features

- High Read parallelism (32 bits)
- Error Correction Code (SEC-DED) to enhance Data Retention
- Sector erase
- RWW is supported between code flash memory and data flash memory modules whereas within single bank RWW is not available. Examples of supported RWW below:
  - CF0 (or CF1) and DFlash
  - CF0 and CF1
- Erase Suspend available (Program Suspend not available)
- Software programmable program/erase protection to avoid unwanted writings

35.3.3 Block diagram

The flash memory module contains one Matrix Module, composed of a single bank: Bank 0, normally used for code storage.

The 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 32 bits wide, while the flash memory registers are on a separate bus 32 bits wide.

The high voltages needed for program/erase operations are internally generated.

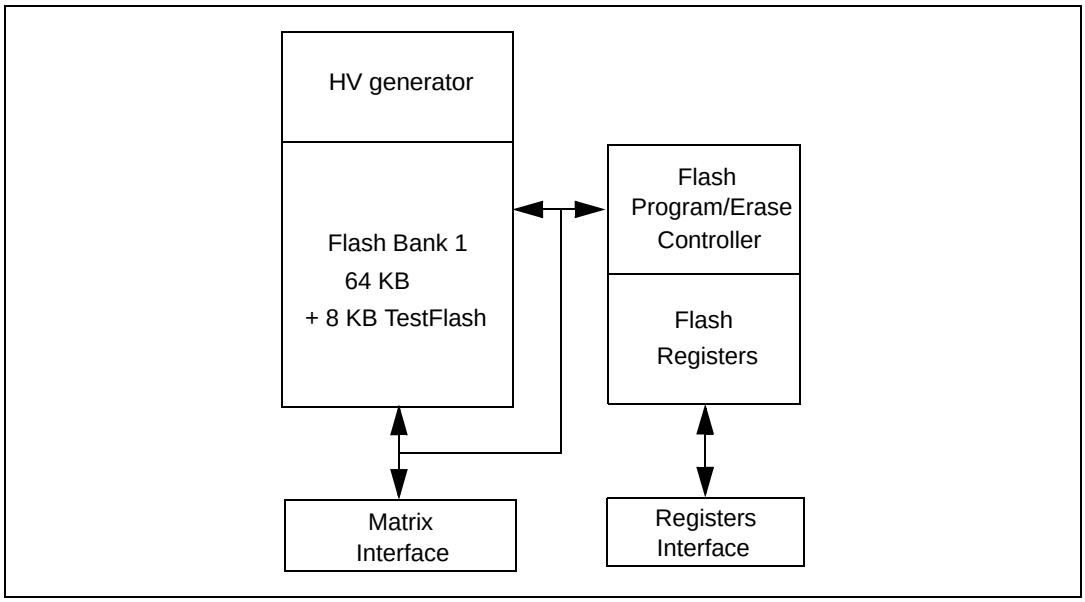


Figure 764. Data flash memory module structure

35.3.4 Functional description

35.3.4.1 Module structure

The data flash memory module is designed for use in embedded device applications which require Data Non-Volatile Memories for EE emulation. The flash memory module is addressable by Word (32 bits) for program and for read.

The flash memory module supports fault tolerance through Error Correction Code (ECC) and/or error detection. 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.

Control logic that works with the software block enables, and software lock mechanisms, is included in the embedded hardware algorithm to guard against accidental program/erase.

The hardware algorithm perform the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to guarantee data integrity and reliability.

A programmed bit in the flash memory module reads as logic level 0 (or low). An erased bit in the flash memory module 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.

Code Flash0 needs to be active for data flash to be active as it is a slave.

35.3.4.2 Data flash memory module sectorization

The data flash memory module supports 64 KB of user memory, plus 8 KB of test memory. There are two User Address Spaces: Low and Mid Address Space. There is only one size of

blocks available to the User in the flash memory Core: 16KB. 8KB is reserved for Test flash memory.

The flash memory module is composed of a single bank (Bank 0): Read-While-Write is not supported. Bank 0 of the 64 KB flash memory 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 flash memory Matrix Module is shown in the *Table 663*.

Table 663. 64 KB data flash memory module sectorization

Bank	Sector	Addresses	Size	Address space
B0	B0F0	0x000000–0x003FFF	16 KB	Low Address Space
B0	B0F1	0x004000–0x007FFF	16 KB	Low Address Space
B0	B0F2	0x008000–0x00BFFF	16 KB	Low Address Space
B0	B0F3	0x00C000–0x00FFFF	16 KB	Low Address Space
B0	Reserved	0x010000–0x07FFFF	—	Reserved
B0	B0TF	0x402000–0x403FFF	8 KB	Test Address Space

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, mid address space against program and erase.

35.3.4.2.1 Test flash memory Block

The TestFlash block exists outside the normal address space and is programmed, erased and read independently of the other blocks. The independent TestFlash block is reserved to store the Non Volatile informations related to Redundancy, Configuration and Protection.

Due to this special usage, the TestFlash sector is not affected by the Column Redundancy. The ECC, on the contrary, is applied also to TestFlash.

The usage of reserved TestFlash sector is detailed in the following table.

Table 664. TestFlash structure

Name	Description	Addresses	Size
	User Reserved	0x403D00 to 0x403DE7	232 byte
NVLML	NV Low/Mid address space block Locking reg	0x403DE8 to 0x403DEF	8 byte
	Reserved	0x403DF0 to 0x403DF7	8 byte
NVSLL	NV Secondary Low/mid add space block Lock reg	0x403DF8 to 0x403DFF	8 byte
	User Reserved	0x403E00 to 0x403EFF	256 byte
	Reserved	0x403F00 to 0x403FB7	184 byte

The Test flash memory block can be enabled by the BIU. When the Test space is enabled, the program operations to the Test block are allowed from 0x403D00 to 0x403EFF (User/Lock area is One Time Programmable). User Mode program of the test block are

enabled only when MCR.PEAS is high. The TestFlash block contains specified data that are needed for the flash memory module or SoC features.

35.3.5 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 and test address space can be read only in the read state.

The flash memory registers are always available for read, also when the module is in disable mode (except few documented registers). 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 MCR.ERS and MCR.ESUS are high and MCR.PGM is low (Erase Suspend).

Notice that no Read-While-Modify is available. flash memory Core reads return 32 bits. Registers reads return 32 bits (1 Word). flash memory Core reads are done through the Bus Interface Unit.

Registers reads to unmapped register address space will return all 0's. Registers writes to unmapped register address space will have no effect. Array reads attempted to invalid locations will result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2n array sizes.

Interlock writes attempted to invalid locations, will result in an interlock occurring, but attempts to program these blocks will not occur since they are forced to be locked. Erase will occur to selected and unlocked blocks even if the interlock write is to an invalid location.

Simultaneous Read cycle on the flash memory Matrix and Read/Write cycles on the Registers are possible. On the contrary Registers Read/Write accesses simultaneous to a flash memory Matrix interlock write are forbidden.

35.3.5.1 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 (MCR.PGM = 1 or MCR.ERS = 1) 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 TEST block or KRAM information, or other inputs, may not be read until MCR.DONE transitions. MCR.DONE may be polled to determine if the flash memory module has transitioned out of reset. Notice that the registers cannot be written until MCR.DONE is high.

35.3.5.2 Power-down mode

The power-down mode allows to turn off all flash memory DC current sources, so that all power dissipation is due only to leakage in this mode.

Reads from or writes to the module are not possible in power-down mode.

The user may not read some registers (UMISR0–1, UT1–1 and part of UT0) until the power-down mode is exited. On the contrary write access is locked on all the registers in Disable 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 set to 1. This means that flash memory module is first put into suspend state (after t_{SUSP}). The User may resume the erase operation at the time the module is enabled by clearing MCR.ESUS bit. MCR.EHV must be high to resume the erase operation.

If the flash memory module is disabled during a program operation, the Disable Mode will be entered only after the programming ends.

35.3.5.3 Slave Mode

It is forbidden to put code flash0 in Disable Mode or in Sleep mode or under reset when the data flash is active.

35.3.6 Register description

The flash memory user registers mapping is shown in the *Table 665*.

Table 665. Data Flash Single Bank Registers

Address offset	Register name	Location
0x0000	*Module Configuration Register (MCR)*	*on page 1201*
0x0004	*Low/Mid address space block Locking register (LML)*	*on page 1206*
0x0008	Reserved	—
0x000C	*Secondary Low/mid address space block Locking register (SLL)*	*on page 1207*
0x0010	*Low/Mid address space block Select register (LMS)*	*on page 1210*
0x0014	Reserved	—
0x0018	*Address Register (ADR)*	*on page 1211*
0x001C-0x0038	Reserved	—
0x003C	*User Test 0 register (UT0)*	*on page 1212*
0x0040	*User Test 1 register (UT1)*	*on page 1214*
0x0044	Reserved	—
0x0048	*User Multiple Input Signature Register 0 (UMISR0)*	*on page 1214*
0x004C	*User Multiple Input Signature Register 1 (UMISR1)*	*on page 1215*
0x0050-0x0058	Reserved	—

Locations 0x0044, 0x0050, 0x0054 and 0x0058 are Write/Read from user point of view but no functionality is associated. Registers are not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

In the following some nonvolatile registers are described. Please notice that such entities are not Flip-Flops, but locations of TestFlash sector with a special meaning.

During the flash memory initialization phase, the FPEC reads these nonvolatile registers and update 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, redundancy, EmbAlgo firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, ...), the volatile registers are filled with all '1's and the flash memory initialization ends setting low the PEG bit of MCR.

Table 666 lists bit access type abbreviations used in this section.

Table 666. Abbreviations

Abbreviation	Case	Description
rw	read/write	The software can read and write to these bits.
rc	read/clear	The software can read and clear to these bits.
r	read-only	The software can only read these bits.
w	write-only	The software should only write to these bits.

35.3.6.1 Module Configuration Register (MCR)

Offset: 0x0000 Reset value: 0x07570X00

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	MAS2	MAS1	MAS0
W	w1c															
Reset	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**

	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

Figure 765. Module Configuration Register (MCR)

The Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Table 667. MCR field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>EDC</td><td>ECC Data Correction (Read/Clear)
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.
Since this bit is an error flag, it must be cleared to ‘0’ by writing 1 to the register location. A write of 0 will have no effect.
0: Reads are occurring normally.
1: An ECC Single Error occurred and was corrected during a previous read.</td></tr>
<tr><td>Bits 1:4</td><td>Reserved (Read Only)
Write these bits has no effect and read these bits always outputs 0.</td></tr>
<tr><td>SIZE[2:0]</td><td>Array Space SIZE 2-0 (Read Only)
The value of SIZE field is dependent upon the size of the flash memory module.
<table><tr><th>SIZE[2:0]</th><th>Array space size</th></tr><tr><td>111</td><td>96 KB (Upper 32 KB is reserved)</td></tr></table></td></tr>
<tr><td>Bit 8</td><td>Reserved (Read Only).
Write this bit has no effect and read this bit always outputs 0.</td></tr>
<tr><td>LAS[2:0]</td><td>Low Address Space 2-0 (Read Only)
The value of the LAS field corresponds to the configuration of the Low Address Space.
<table><tr><th>LAS[2:0]</th><th>Low address space sectorization</th></tr><tr><td>101</td><td>6 x 16 KB (Upper 32 KB is reserved)</td></tr></table></td></tr>
<tr><td>Bit 12</td><td>Reserved (Read Only)
Write these bits has no effect and read these bits always outputs 0.</td></tr>
<tr><td>MAS2-0</td><td>MAS2-0: Mid Address Space (Read Only)
The value of the MAS field corresponds to the configuration of the Mid Address Space.
<table><tr><th>MAS</th><th>Mid address space sectorization</th></tr><tr><td>111</td><td>MID not present</td></tr></table></td></tr>
</table>

Table 667. MCR field descriptions (continued)

Field	Description
EER	*ECC event Error* (Read/Clear) 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. Since this bit is an error flag, it must be cleared to ‘0’ by writing 1 to the register location. A write of 0 will have no effect. 0: Reads are occurring normally. 1: An ECC Double Error occurred during a previous read.
RWE	*Read-while-Write event Error* (Read/Clear) 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. Since this bit is an error flag, it must be cleared to ‘0’ by writing 1 to the register location. A write of 0 will have no effect. 0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.
Bits 18:19	*Reserved* (Read Only) Write these bits has no effect and read these bits always outputs 0.
PEAS	*Program/Erase Access Space* (Read Only) PEAS is used to indicate which space is valid for program and erase operations: main array space or 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 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: Test address space is disabled for program/erase and main address space enabled. 1: Test address space is enabled for program/erase and main address space disabled.
DONE	*modify operation DONE* (Read Only) 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 667. MCR field descriptions (continued)

Field	Description
PEG	*Program/Erase Good* (Read Only) The PEG bit indicates the completion status of the last flash memory Program, Erase, AIC or MM sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the Program, Erase, AIC or MM high voltage operations. Aborting a Program/Erase/AIC/MM 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/AIC/MM 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 AIC or MM 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. 0: Program or Erase, operation failed or aborted. 1: Program or Erase operation successful. 0: AIC or MM aborted. 1: AIC or MM operation successfully concluded, with or without checksum errors.
Bits 23:26	*Reserved* (Read Only) Write these bits has no effect and read these bits always outputs 0.
PGM	*ProGraM* (Read/Write) 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* (Read/Write) Write this bit has no effect, but the written data can be read back.
ERS	*ERaSe* (Read/Write) 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.

Table 667. MCR field descriptions (continued)

Field	Description
ESUS	*Erase SUSpend* (Read/Write) 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* (Read/Write) 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 undeterminate 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.

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash 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 668*.

Table 668. MCR bits set/clear priority levels

Priority Level	MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more MCR bits simultaneously then only the bit with the lowest priority level is written.

35.3.6.2 Low/Mid address space block Locking register (LML)

Address offset: 0x0004

Reset value: 0x00X0_00XX, initially determined by NVLML value from test sector.

35.3.6.2.1 Nonvolatile Low/Mid address space block Locking register (NVLML)

Offset: 0x403DE8 Delivery value: 0xFFFF_FFFF

	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	0	0	0	0	0	0	0	0	0	0	0	X	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	LLK3	LLK2	LLK1	LLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	X	X	X	X

Figure 766. Nonvolatile Low/Mid address space block Locking register (NVLML)

The Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the SLL register, determine if the block is locked from Program or Erase. An “OR” of LML and SLL determine the final lock status.

The LML register has a related Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for LML: the NVLML register content is read during the reset phase of the flash memory module, and loaded into the LML.

Table 669. LML field descriptions

Field	Description
LME	*Low/Mid address space block Enable* (Read Only) This bit is used to enable the Lock registers (TSLK and LLK5-0) 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 LML register. 0: Low Address Locks are disabled: TSLK and LLK5-0 cannot be written. 1: Low Address Locks are enabled: TSLK and LLK5-0 can be written.
Bits 1:10	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.

Table 669. LML field descriptions (continued)

Field	Description
TSLK	*Test address space block LocK* (Read/Write) This bit is used to lock the block of Test 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 block is locked for Program and Erase. A value of 0 in the TSLK register signifies that the Test block is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended or if a margin mode is on going. 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 Address Space Block is unlocked and can be modified (also if SLL.STSLK = 0). 1: Test Address Space Block is locked and cannot be modified.
Bits 12:25	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
LLK[3:0]	*Low address space block LocK 3-0* (Read/Write) These bits are used to lock the blocks of Low Address Space from Program and Erase. LLK3:0 are related to sectors B0F3-0, respectively. 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 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. LLK is not writable unless LME is high. 0: Low Address Space Block is unlocked and can be modified (also if SLL.SLK = 0). 1: Low Address Space Block is locked and cannot be modified.

35.3.6.3 Secondary Low/mid address space block Locking register (SLL)

Address offset: 0x000C

Reset value: 0x00X0_00XX

35.3.6.3.1 Nonvolatile Secondary Low/mid address space block Locking register (NVSLL)

Offset: 0x403DF8 Delivery value: 0xFFFF_FFFF

	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												rw/X				
Reset	0	0	0	0	0	0	0	0	0	0	0	X	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	SLK3	SLK2	SLK1	SLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	X	X	X	X

Figure 767. Nonvolatile Secondary Low/mid address space block Locking reg (NVSLL)

The Secondary Low/Mid Address Space Block Locking register provides an alternative means to protect blocks from being modified. These bits, along with bits in the LML register, determine if the block is locked from Program or Erase. An “OR” of LML and SLL determine the final lock status.

The 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 NVSLL register content is read and loaded into the SLL.

Table 670. SLL field descriptions

Field	Description
SLE	*Secondary Low/mid address space block Enable* (Read Only) This bit is used to enable the Lock registers (STSLK and SLK 5-0) 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 SLL register. 0: Secondary Low/Mid Address Locks are disabled: STSLK and SLK 5-0 cannot be written. 1: Secondary Low/Mid Address Locks are enabled: STSLK and SLK 5-0 can be written.
Bits 1:10	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.

Table 670. SLL field descriptions (continued)

Field	Description
STSLK	*Secondary Test/Shadow address space block LocK* (Read/Write) This bit is used as an alternate means to lock the block of Test 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 block is locked for Program and Erase. A value of 0 in the STSLK register signifies that the Test block is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended or if a margin mode is on going. 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 Address Space Block is unlocked and can be modified (also if LML.TSLK = 0). 1: Test Address Space Block is locked and cannot be modified.
Bits 12:25	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
SLK[3:0]	*Secondary Low address space block locK 3-0* (Read/Write) These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase. SLK[35:0] are related to sectors B0F3-0, respectively. A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses. The SLK register is not writable once an interlock write is completed until MCR.DONE is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset 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. SLK is not writable unless SLE is high. 0: Low Address Space Block is unlocked and can be modified (also if LML.LLK = 0). 1: Low Address Space Block is locked and cannot be modified.

35.3.6.4 Low/Mid address space block Select register (LMS)

Offset: 0x00010 Reset value: 0x0000_0000

	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	LSL3	LSL2	LSL1	LSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

Figure 768. Low/Mid address space block Select register (LMS)

The Low/Mid Address Space Block Select register provides a means to select blocks to be operated on during erase.

Table 671. LMS field descriptions

Field	Description
Bits 0:25	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
LSL[3:0]	*Low address space block SeLect 3-0* (Read/Write) 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 B0F3-0, 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 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. 0: Low Address Space Block is unselected for Erase. 1: Low Address Space Block is selected for Erase.

35.3.6.5 Address Register (ADR)

Offset: 0x00018 Reset value: 0x0000_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	AD[22:16]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD[15:2]														0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 769. Address Register (ADR)

The Address Register 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.

Table 672. ADR field descriptions

Field	Description
AD[22:2]	*ADdress 22-2* (Read Only) The Address Register provides the first failing address in the event of ECC error (MCR.EER set) or the first failing address in the event of RWW error (MCR.RWE set), or the address of a failure that may have occurred in a FPEC operation (MCR.PEG cleared). The Address Register also provides the first address at which an ECC single error correction occurs (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 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 673*. In User Mode the Address Register is read only.

Table 673. ADR content: priority list

Priority Level	Error Flag	ADR content
1	MCR.EER = 1	Address of first ECC Double Error
2	MCR.RWE = 1	Address of first RWW Error
3	MCR.PEG = 0	Address of first FPEC Error
4	MCR.EDC = 1	Address of first ECC Single Error Correction

35.3.6.6 User Test 0 register (UT0)

Offset: 0x0003C Reset value: 0x0000_0001

	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	0	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W																
Reset	0	0	0	0	0	0	0	0	0	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

Figure 770. User Test 0 register (UT0)

The User Test feature gives the User of the flash memory module the ability to perform test features on the flash memory. 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 DSI6-0 of the User Test 0 Register are not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 674. UT0 field descriptions

Field	Description
UTE	*User Test Enable* (Read/Clear) This status bit gives indication when User Test is enabled. All bits in UT0-1 and UMISR0-1 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 UT0 register.
Bits 1:8	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
DSI[6:0]	*Data Syndrome Input 6-0* (Read/Write) These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. Bits DSI[6:0] correspond to the 7 syndrome bits on a single word. These bits are not accessible whenever MCR.DONE or 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.
Bits 16:24	*Reserved* (Read Only). Write these bits has no effect and read these bits always outputs 0.
Bit 25	*Reserved* (Read/Write). This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 674. UT0 field descriptions (continued)

Field	Description
MRE	*Margin Read Enable* (Read/Write) MRE enables margin reads to be done. This bit, combined with MRV, enables start of FPEC margin reads respect to erased or programmed value. Outputs of margin read are: checksum values in UMISR0-1. This bit is not accessible whenever MCR.DONE or 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* (Read/Write) If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever MCR.DONE or UT0.AID are low: 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* (Read/Write) EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever MCR.DONE or 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* (Read/Write) AIS determines the address sequence to be used during array integrity checks. The default sequence (AIS=0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS=1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. This bit is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect. 0: Array Integrity sequence is proprietary sequence. 1: Array Integrity sequence is sequential.
AIE	*Array Integrity Enable* (Read/Write) AIE set to 1 starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (UMISR0-1) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if MCR.ERS, MCR.PGM and MCR.EHV are all low. 0: Array Integrity Checks are not enabled. 1: Array Integrity Checks are enabled.
AID	*Array Integrity Done* (Read Only) 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 (UMISR0-1) can be checked. 0: Array Integrity Check is on-going. 1: Array Integrity Check is done.

35.3.6.7 User Test 1 register (UT1)

Offset: 0x00040 Reset value: 0x0000_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
Reset	0	0	0	0	0	0	0	0	0	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	DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 771. User Test 1 register (UT1)

The User Test 1 Register allows to enable the checks on the ECC logic related to the Word. The User Test 1 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 675. UT1 field descriptions

Field	Description
DAI[31:00]	*Data Array Input 31-0* (Read/Write) 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 word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

35.3.6.8 User Multiple Input Signature Register 0 (UMISR0)

Offset: 0x00048 Reset value: 0x0000_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS31	MS30	MS29	MS28	MS27	MS26	MS25	MS24	MS23	MS22	MS21	MS20	MS19	MS18	MS17	MS16
Reset	0	0	0	0	0	0	0	0	0	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	MS15	MS14	MS13	MS12	MS11	MS10	MS09	MS08	MS07	MS06	MS05	MS04	MS03	MS02	MS01	MS00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 772. User Multiple Input Signature Register 0 (UMISR0)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity. The User Multiple Input Signature Register 0 represent the bits 31-0 of the word. The UMISR0 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 676. UMSIR0 field descriptions

Field	Description
MS[31:00]	*Multiple input Signature 31-00* (Read/Write) 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 UMISR0 register.

35.3.6.9 User Multiple Input Signature Register 1 (UMISR1)

Offset: 0x0004C Reset value: 0x0000_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS63	MS62	MS61	MS60	MS59	MS58	MS57	MS56	MS55	MS54	MS53	MS52	MS51	MS50	MS49	MS48
Reset	0	0	0	0	0	0	0	0	0	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	MS47	MS46	MS45	MS44	MS43	MS42	MS41	MS40	MS39	MS38	MS37	MS36	MS35	MS34	MS33	MS32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 773. User Multiple Input Signature Register 1 (UMISR1)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity. The User Multiple Input Signature Register 1 represent the ECC bits of the 32 bits word: bits 6-0 are the ECC bits for the Word; bits 10 and 11 of MISR are respectively the double and single ECC error detection. The UMISR1 Register is not accessible whenever MCR.DONE or UT0.AID are low: reading returns indeterminate data while writing has no effect.

Table 677. UMISR1 field descriptions

Field	Description
MS[63:32]	*Multiple input Signature 63-32* (Read/Write) These bits represents the MISR value obtained accumulating: 7 ECC bits for the Word (on MS38-32); single ECC error detection (on MS42); double ECC error detection (on MS43); The MS can be seeded to any value by writing the UMISR1 register.

35.3.7 Programming considerations

35.3.7.1 Modify operation

All the 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-Modify is not supported).

During a flash memory Modify Operation any attempt to read any flash memory location will output invalid data and bit RWE of MCR will be automatically set. This means that the flash memory module is not fetchable when a Modify Operation is active: the Modify Operation commands must be executed from another Memory (internal Ram or external Memory). If during a Modify Operation a reset occurs, the operation is suddenly terminated and the module is reset to Read Mode. The data integrity of the flash memory section where the Modify Operation has been terminated or aborted is not guaranteed: the interrupted flash memory Modify Operation must be repeated. In general each Modify Operation is started through a sequence of 3 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 EHV in MCR or AIE in UT0. 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 the *Table 661*.

Table 678. 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 bit MCR.EHV (or UT0.AIE) is set, all the operands can no more be modified until bit MCR.DONE (or UT0.AID) is high.

In general each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for bit MCR.DONE (or UT0.AID) to go high.
2. Check operation result: check bit MCR.PEG (or compare UMISR0-1 with expected value).
3. Switch off FPEC by resetting MCR.EHV (or UT0.AIE).
4. Deselect current operation by clearing MCR.PGM/ERS (or UT0.MRE/EIE).

In the following all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

35.3.7.2 Word program

A flash memory program sequence operates on any word within the flash memory core.

Whenever flash bits are programmed, ECC bits also get programmed, unless the selected address belongs to a sector in which the ECC has been disabled in order to allow bit manipulation. ECC is handled on a 32-bit boundary.

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 words within a single program sequence.

The Program operation consists of the following sequence of events:

1. Change the value in the MCR.PGM bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.
   a) Write the first address to be programmed with the program data.
   b) The flash memory module latches address bits (22:2) at this time.
   c) The flash memory module latches data written as well.
   d) This write is referred to as a program data interlock write. An interlock is at 32 bits.
3. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 8 to terminate.
4. Wait until the MCR[DONE] bit goes high.
5. Confirm MCR[PEG]=1.
6. Write a logic 0 to the MCR[EHV] bit.
7. If more addresses are to be programmed, return to step 2.
8. 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 test or normal array space will be programmed by causing MCR[PEAS] to be set/cleared. An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV]. After the interlock write, additional writes only affect the data to be programmed in the word. 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 7 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 8 Word program of data 0x55AA55AA at address 0x00AAA8

```
MCR   = 0x00000010;/* Set PGM in MCR: Select Operation */
(0x00AAA8)= 0x55AA55AA;/* Latch Address and 32 LSB 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 */
```

35.3.7.3 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). 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 register(s). 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 9 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 */
```

35.3.7.3.1 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 tESUS 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 10 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 11 Sector erase resume

```
MCR   = 0x00000005;/* Reset ESUS in MCR: Erase Resume */
```

35.3.7.4 User Test Mode

User Test Mode is a procedure to check the integrity of the flash memory module.

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 (RWE of MCR set).

It is not allowed to perform User Test operations on the Test and Shadow blocks.

35.3.7.4.1 Array integrity self check

Array Integrity is checked using a pre-defined 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 UMISR0-1), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32 bit register. The 32 bit data, the 7 ECC data and the single and double ECC errors of the Word are therefore captured by the MISR through 2 different read accesses at the same location. The whole check is done through 2 complete scans of the memory address space:

1. The 1st pass will scan only bits 31-0 of each word.
2. The 2nd pass will scan only the ECC bits (7) and the single and double ECC errors (1 + 1) of each word.

The 32 data bit and the 7 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 UMISR0-1 registers.

Once command is started, Array Integrity check is run by FPEC using system clock and the number of wait states identified by address and data wait states.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1's to the appropriate register(s) in LMS. 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. Clear (or insert seed) UMISR0-1
5. Write a logic 1 to the UT0.AIE bit to start the Array Integrity Check.
6. Wait until the UT0.AID bit goes high.
7. Compare UMISR0-1 content with the expected result.
8. Write a logic 0 to the UT0.AIE bit.
9. If more blocks are to be checked, return to step 2.
10. clear UT0 writing UT0.UTE to '0'

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. 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 12 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*/
UT0   = 0x00000000;/* Reset UTE and AIE in UT0: Operation End */
```

35.3.7.4.2 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 UMISR0-1. Since Margin reads are done at voltages that differ than the normal read voltage, lifetime expectancy of the flash memory module 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. It is recommended the Margin reads be done on a limited basis (less than 10 times before the next chip erase).

The Margin Read Setup operation consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1’s to the appropriate register(s) in LMS. Note that Lock and Select are independent. If a block is selected and locked, no Margin Read will occur.
3. Set eventually UT0.AIS bit for a sequential addressing only.
4. Change the value in the UT0.MRE bit from 0 to 1.
5. Select the Margin level: UT0.MRV=0 for 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 UMISR0-1 content with the expected result.
9. Write a logic 0 to the UT0.AIE UT0.MRE and UT0.MRV bits.

It is recommended to leave UT0.AIS at 1 and use the linear address sequence and takes less time. While UT0.AID is low and UT0.AIE is high, the User may clear AIE, resulting in a Margin Mode abort. UT0.AID must be checked to know when the aborting command has completed.

Example 13 Margin read setup versus ‘1’s

```
UT0   = 0xF9F99999;/* Set UTE in UT0: Enable User Test */
UT0   = 0x80000020;/* Set MRE in UT0: Select Operation */
UT0   = 0x80000030;/* Set MRV in UT0: Select Margin versus 1’s */
UT0   = 0x80000032;/* 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*/
UT0   = 0x00000000;/* Reset UTE, AIE, MRE, MRV in UT0: Deselect Operation */
```

35.3.7.4.3 ECC logic check

ECC logic can be checked by forcing the input of ECC logic: the 32 bits of data and the 7 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the word.

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 UTE in UT0 by writing the related password in UT0.
2. Write in UT1.DAI31-0 Word Input value.
3. Write in UT0.DSI6-0 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 UMISR0-1 content with the expected result.
8. Write a logic 0 to the UT0.AIE bit.

Notice that when UT0.AID is low UMISR0-1, UT1 and bits MRE, MRV, EIE, AIS and DSI6-0 of UT0 are not accessible: reading returns undeterminate data and write has no effect.

Example 14 ECC logic check

```
UT0   = 0xF9F99999;/* Set UTE in UT0: Enable User Test */
UT1   = 0x55555555;/* Set DAI31-0 in UT1: Word Input Data */
UT0   = 0x80380000;/* Set DSI6-0 in UT0: Syndrome Input Data */
UT0   = 0x80380008;/* Set EIE in UT0: Select ECC Logic Check */
UT0   = 0x8038000A;/* 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) */
UT0   = 0x00000000;/* Reset UTE, AIE and EIE in UT0: Operation End */
```

35.3.8 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. 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. Word size is fixed at 32 bits.

At each Word of 32 bits there are associated 7 ECC bits that are programmed in such a way to guarantee a Single Error Correction and a Double Error Detection (SEC-DED).

35.3.8.1 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.

35.3.8.2 ECC Algorithms Features

The flash memory module ECC Algorithm supports the following features:

- All ‘0’s Error
  - The All ‘0’s Error Algorithm detects as Double ECC Error any Word in which all the 39 bits are “0’s.
- All ‘1’s No Error
  - The All ‘1’s No Error Algorithm detects as valid any Word read on a just erased sector (all the 39 bits are “1’s).
    This option allows to perform a Blank Check after a Sector Erase operation.
- Bit Manipulation
  - 8 bits clears (by byte) are allowed on any erased word maintaining valid the syndrome of the word. 8 bits clears can be done on any byte of the word without a specific order. This featured is intended as a counter for EE-Emulation.
    Example 1: data patterns with the same ECC syndrome (equal to 0x7F).

```
0xFFFFFFFF -> 7F
0xFFFFFF00 -> 7F
0xFFFF00FF -> 7F
0xFF00FFFF -> 7F
0x00FFFFFF -> 7F
0xFFFF0000 -> 7F
0x0000FFFF -> 7F
0xFF000000 -> 7F
0x000000FF -> 7F
0x00000000 -> 7F
```

- Enhanced flagging
  In case flagging method is required for more then 4 writes, the following sequence allows up to 7 pattern with the same ECC syndrome.

```
0xFFFFFFFF -> 7F
0xFFFFFFB1 -> 7F
0xFFFFFF00 -> 7F
0xFFACFF00 -> 7F
0xFF00FF00 -> 7F
0xCA00FF00 -> 7F
0x0000FF00 -> 7F
0x00000000 -> 7F
```

- 3 Bits Error Detection
  - 40.21% of the possible 3 bits errors are detects as Double ECC Error.
  - 59.79% of the possible 3 bits errors are instead detects as Single ECC Error and miscorrected.

35.3.9 Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in flash memory sectors. The Censored Mode to avoid piracy must be managed by the associated Code flash memory module embedded in the same device.

35.3.9.1 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 they are stored in volatile registers that act as actuators.

The reset state of all the Volatile Modify Protection Registers is the protected state.

All the nonvolatile Modify Protection registers can be programmed through a normal 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 Address Space Block against program and erase.

Software locking is done through the LML (Low/Mid Address Space Block Lock Register). An alternate means to enable software locking for blocks of Low Address Space only is through the SLL (Secondary Low/Mid Address Space Block Lock Register).

All these registers have a Non Volatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash Non Volatile image is at all 1’s that means all sectors locked. By programming the Non Volatile locations in TestFlash the selected sectors can be unlocked. Being the TestFlash One Time Programmable (i.e. 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.

35.4 Platform Flash Controller

35.4.1 Introduction

This section provides an introduction to the 2-port platform flash controller for Standard Product Platforms (SPP).The platform flash controller acts as the interface between two system bus masters (AHB-Lite 2.v6) and up to three banks of flash memory arrays. It intelligently converts the protocols between the system bus ports and the dedicated flash array interfaces.

Throughout this document, several important terms are used to describe the platform flash 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 controller. This flash controller supports 2 AHB ports. For the platform design, *platform flash controller* port 0 (*P0) is always connected to the e200z4d instruction set, platform flash controller* port 1 (*P1) is always connected to the e200z4d data set and all other masters including e200z0h.*
- **Bank** — This term is used to describe the attached flash memories. From the platform flash controller’s perspective, there are three attached banks of flash memory. There are two “code flash” arrays required and they are attached to banks 0 and 2. The platform flash controller treats banks 0 and 2 in a common manner with various

configuration fields of the programming model shared across the two banks. Additionally, there is a “data flash” attached to bank1.

- **Array** — Within each memory bank, there are one (or more) flash array instantiations. Regardless of the number of array instantiations or the number of populated banks, the operating configuration of the platform flash controller is defined by the register values contained in bank0 array0.
- **Page** — This value defines the number of bits read from the flash array in a single access. For the code flash, the page size is 128-bits (16 bytes). The data flash has a page size of 32-bits (4 bytes).

The nomenclature “page buffers and “line buffers” are used interchangeably.

From an architectural and programming model perspective, there are two configuration registers associated with the platform flash controller. These variables define the 2 AHB input ports (p0 and p1) initiating transactions and the three destination flash memory banks (b0, b1, b2). The following abbreviations for these variables are used throughout the document:

p0	AHB port 0
p1	AHB port 1
b0, bk0	flash memory bank0
b1, bk1	flash memory bank1
b2, bk2	flash memory bank2
b02	flash memory banks 0 and 2

Finally since the page buffers and temporary holding registers are associated with *both* an AHB input port and a flash bank, they use a *bx_py* nomenclature. For example, the b0_p0 page buffer refers to the bank0, port 0 storage elements.

35.4.1.1 Overview

The platform flash controller supports a 64-bit data bus width at the two AHB ports and connections to two 128-bit read data interfaces from each of the code flash memory banks and a 32-bit read data interface from the data flash bank, where each bank contains one (or more) instantiations of the flash memory array. Flash bank0 is connected to the first code flash memory, bank2 is connected to a second code flash memory, and bank1 is connected to the data flash memory. The memory controller capabilities vary between the banks with each bank’s functionality optimized for the typical use cases associated with the attached flash memory.

As an example, the platform flash controller logic associated with bank0 contains 2 four-entry 128-bit page buffers, one for each AHB port which pre-fetches sequential lines of data from the flash array into the buffer. This structure is repeated for bank2, providing a total of four copies of the 4-entry page buffer between bank 0 and 2. The controller logic associated with bank1 is simpler and supports two 64-bit registers (one for each AHB port) which serve as temporary page holding registers and no support of any prefetching. Prefetch buffer hits from any of the page buffers or temporary holding registers support zero-wait AHB data phase responses. AHB read requests which miss the buffers generate the needed flash array access and the read data is forwarded to the AHB port upon completion, typically incurring 5 wait states for C-flash (bank 0/2) and 13 wait states for D-flash (bank 1) at an operating frequency of 120 MHz. *The logic of the platform flash controller is structured to support simultaneous AHB accesses from the two ports fully in parallel when the references are targeted to different memory banks.* If simultaneous AHB port accesses reference the same bank, then arbitration logic within the platform flash controller determines the order the references are granted access to the bank.

35.4.1.2 Features

The following list summarizes the key features of the platform flash controller:

- Dual AHB input port interfaces support a 64-bit data bus. All AHB aligned and unaligned reads within the 64-bit container are supported. Only aligned word writes are supported.
- Code flash array interfaces support a 128-bit read data bus and 64-bit write data bus. Data flash array interface supports a 32-bit read data bus and write data bus.
- Internal hardware structure supports fully concurrent accesses from the dual AHB input ports when accessing different flash banks
  - If the AHB ports reference the same flash bank, there is arbitration logic which determines the order the accesses are granted access to the bank
  - Programmable arbitration allows the user to select fixed priority or round-robin
- Total flash page storage in the platform flash controller includes four 4-entry page buffers (b0_p0, b0_p1, b2_p0, b2_p1) and two 64-bit temporary holding registers (b1_p0, b1_p1).
  - Each AHB input port provides configurable and independent read buffering and page prefetch support for banks 0 and 2
  - Each AHB input port includes four page read buffers (each 128 bits wide) and a prefetch controller 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.
  - Each AHB input port interfaces to the data flash (bank1) includes a 64-bit register to temporarily hold up to two data flash pages. 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 operation abort, and optional abort notification interrupt
- Separate and independent configurable access timing (common settings for banks 0 and 2, separate settings for bank1)
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit flash ECC events
- Typical operating configuration loaded into programming model by system reset

35.4.1.3 Modes of Operation

The platform flash controller module does not support any special modes of operation. Its operation is driven from the AMBA-AHB memory references it receives from the platform's bus masters. Its configuration is defined by the setting of its programming model registers, physically located as part of the flash array modules.

35.4.2 External Signal Descriptions

The platform flash controller does not directly interface with any external signals. Its primary internal interfaces include two input connections from AMBA-AHB crossbar (or memory protection unit) slave ports and output connections with three banks (2 code and 1 data) of flash memory, each containing one or more instantiations of the flash array. Additionally, the

operating configuration for the platform flash controller is defined by the contents of bank0 array0 registers which are inputs to the module.

A summary of the platform flash controller internal connections is shown in *Table 679*.

Table 679. platform flash controller Module Connections

platform flash controller Connection	Description
Input p0	e200z4d Instruction Set
Input p1	e200z4d data set plus all other masters including e200z0h
Output b0	Bank0, Code Flash 0
Output b1	Bank1, Data Flash
Output b2	Bank2, Code Flash 1

35.4.3 Memory map and register description

There are two memory maps associated with the platform flash controller: one for the flash memory space and another for the program-visible control and configuration registers. The flash memory space is accessed via the AMBA-AHB ports while the program-visible registers are accessed via the slave peripheral bus.

There are no program-visible registers that physically reside inside the platform flash controller. Rather, the platform flash controller receives control and configuration information from the flash array controller(s) to determine the operating configuration. These are part of the flash array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

Note: Updating the configuration fields that control the platform flash controller behavior should only occur while the flash controller is idle. Changing configuration settings while a flash access is in progress can lead to non-deterministic behavior.

First, consider the flash memory space accessed via transactions from the platform flash controller's AHB ports. To support the three separate flash memory banks, the platform flash controller decodes the system address of the memory request to steer the access to the appropriate memory bank. The program-visible control and configuration registers associated with each memory array are included in the slave peripheral address region. See *Table 680*.

Table 680. Flash-related regions in the system memory map

Start Address	End Address	BlockSize [KB]	Region
0x0000_0000	0x0007_7FFF	512	Code flash0 array 0
0x0008_0000	0x000F_FFFF	512	Code flash0 array 1
0x0010_0000	0x0017_FFFF	512	Code flash0 array 2
0x0018_0000	0x001F_FFFF	512	Code flash1 array 0
0x0020_0000	0x0027_FFFF	512	Code flash1 array 1
0x0028_0000	0x002F_FFFF	512	Code flash1 array 2
0x0030_0000	0x007F_FFFF	5120	Reserved

Table 680. Flash-related regions in the system memory map (continued)

Start Address	End Address	BlockSize [KB]	Region
0x0080_0000	0x0080_FFFF	64	Data flash array 0
0x0081_0000	0x00E0_7FFF	6256	Reserved
0x00E0_8000	0x00E0_BFFF	16	Code flash array 1: test sector
0x00E0_C000	0x00FF_BFFF	2026	Reserved
0x00FF_C000	0x00FF_FFFF	16	Code flash array 0: shadow sector
0x0100_0000	0x1FFF_FFFF	507904	Emulation Mapping
0xFFE8_8000	0xFFE8_BFFF	16	Code flash array 0 configuration(1)
0xFFE8_C000	0xFFE8_FFFF	16	Data flash array 0 configuration(1)
0xFFEB_0000	0xFFEB_3FFF	16	Code flash array 1 configuration(2)

1. This region is also aliased to address 0xC3F8_nnnn.
2. This region is also aliased to address 0xC3FB_nnnn.

Notice the sector orientation for Code Flash1 maps the LAS/MAS regions (512 KB) to the end of logical system address range for Code Flash1. Accordingly, the HAS region (1 MB) is mapped to logical start of the system address range for Code Flash1. Additional decoding is required during program and erase to ensure the corresponding physical sectors are selected correctly for a given system address.

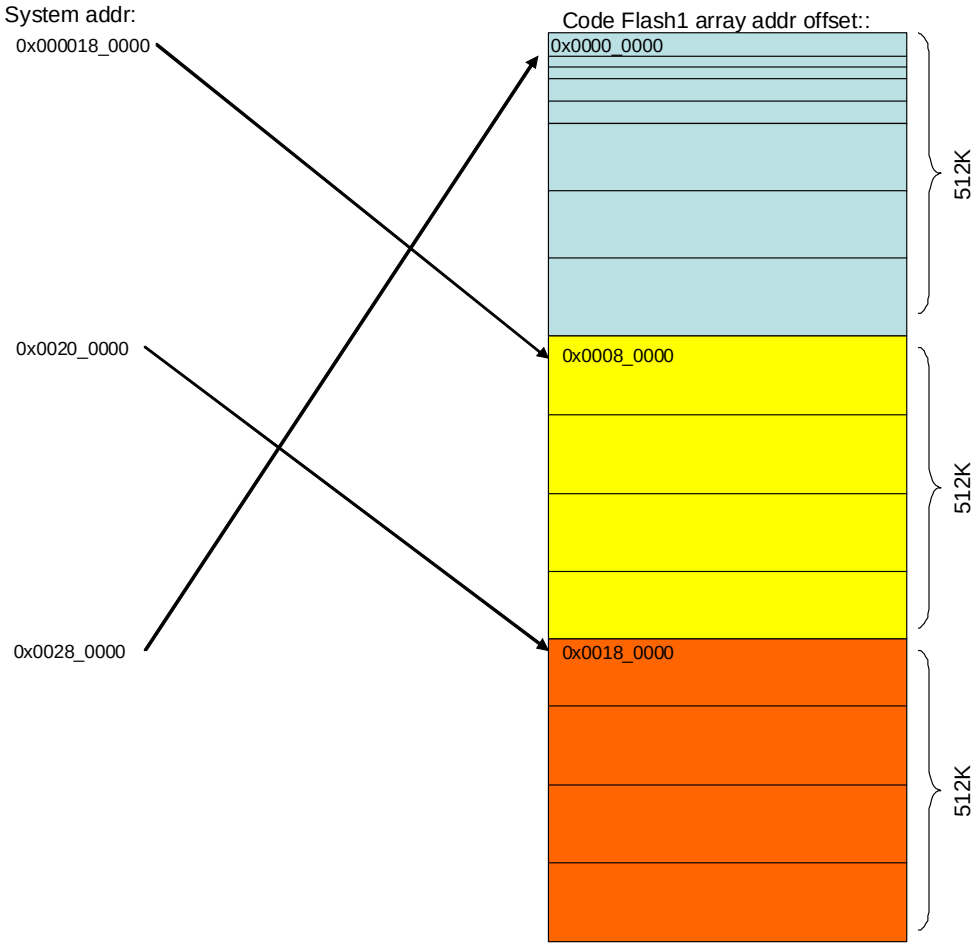


Figure 774. Code Flash1 Array Address Decode

For additional information on the address-based read access timing for emulation of other memory types, see *Section 35.4.4.14: Wait-State Emulation*.

Next, consider the memory map associated with the control and configuration registers.

There are multiple registers that control operation of the platform flash controller. These registers are generically defined as “Bus Interface Unit *n* (BIU *n*) Register” in the flash array documentation, where *n* = 0, 1, 2, 3 and are to be only referenced with 32-bit accesses. Note the first two flash array registers (BIU0, BIU1) are reset to an SoC-defined value, while the remaining two array registers (BIU2, BIU3) are loaded at reset from specific locations in the array’s shadow region.

Regardless of the number of populated banks or the number of flash arrays included in a given bank, *the configuration of the platform flash controller is wholly specified by the BIU registers associated with bank0 array0*. These register settings define the operating behavior of **all** flash banks; it is recommended that the BIU registers for all physically-present arrays be set to the bank0 array0 values.

Note: To perform program and erase operations, the control registers in the actual referenced flash array must be programmed, but the configuration of the platform flash controller module is defined by the BIUn registers of bank0 array0.

The 32-bit memory map for the platform flash controller control registers are shown in *Table 681*.

Table 681. platform flash controller 32-bit Memory Map

Address Offset	Register	Location
0x01C	Platform Flash Configuration Register 0 (PFCR0)	*on page 1231*
0x020	Platform Flash Configuration Register 1 (PFCR1)	*on page 1234*
0x024	Platform Flash Access Protection Register (PFAPR)	*on page 1237*

Within the platform flash controller's programming model, there are a variety of control and configuration fields. Some are associated with the operating configuration of the memory banks, while others are related to the behavior of the AHB master ports.

Due to limitations in the available register bits in the programming model, the PFLASH controllers do *not* provide completely symmetric capabilities for the various memory banks. In fact, the platform flash controller groups together the attributes of the two code flash arrays attached to bank0 and bank2 of the controller while the configuration of the data flash (bank1) is treated separately.

First, consider the operating configuration of the flash banks. In particular, there are 4 unique configuration fields that are associated with a bank. These include all the parameters associated with the timing (read and write wait states, address pipeline control) as well as the read-while-write control field. Accordingly, the programming model supports two separate sets of these 4 fields: one for banks 0 and 2 in PFCR0, and another for bank1 in PFCR1:

// ***per memory bank configuration controls***

b02_apc, b1_apc// address pipeline control

b02_rwsc, b1_rwsc// read wait state control

b02_rwwc, b1_rwwc// read-while-write control

where *b02* is used to refer to configuration and control information common to banks 0 and 2 while *b1* refers to bank1.

Second, there are a total of 6 configuration fields that relate to the operation of the platform flash controller's page buffers. These fields are defined on a "per port" basis since the control needs to be associated *with the AHB master port* and *not the destination flash bank*. In addition, recall that bank1, connected to the data flash, does not support prefetching, so the configuration controls for that bank are considerably reduced compared to banks 0 and 2. The resulting fields are:

// ***per ahb master port configuration controls***

b02_p0_bcfg, b02_p1_bcfg// page buffer configuration

b02_p0_dpfen, b02_p1_dpfen// data prefetch enable

b02_p0_ipfen, b02_p1_ipfen// inst prefetch enable

b02_p0_pflim, b02_p1_pflim// page buffer prefetch limit

b02_p0_bfe, b02_p1_bfe// page buffer enable for banks 0,2

b1_p0_bfe, b1_p1_bfe// page buffer enable for bank1

All these fields are located in the PFCR0 and PFCR1 registers described below.

35.4.3.1 Platform Flash Configuration Register 0 (PFCR0)

This register defines the configuration associated with flash memory banks 0 and 2. Collectively, this corresponds to the “code flash” and the operating configuration defined by certain fields applies to both memory banks. Additionally, it includes fields that provide specific information for the two separate AHB ports (p0 and p1). This register should only be written with 32-bit write operations to avoid any issues associated with register "incoherency" caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

Note: This register should not be updated directly from flash memory. Before modifying the wait states:

1. Transfer code execution to RAM.

2. Modify this register.

3. Return to the execution from flash memory.

Offset 0x01c Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	B02_APC					0	0	0	0	0	B02_RWSC					B02_RWWC2
W																
Reset	0	0	1	0	1	0	0	0	0	0	0	0	1	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B02_RWWC1	B02_P1_BCFG		B02_P1_DPFE	B02_P1_IPFE	B02_P1_PFLM		B02_P1_BFE	B02_RWWC0	B0_P0_BCFG		B02_P0_DPFE	B02_P0_IPFE	B02_P0_PFLM		B02_P0_BFE
W																
Reset	1	1	1	0	1	1	0	1	1	1	1	0	1	1	0	1

Figure 775. PFLASH Configuration Register 0 (PFCR0)

Table 682. PFCR0 field descriptions

Field	Description
B02_APC	Bank0+2 Address Pipelining Control. This field is used to control the number of cycles between flash array access requests. This field must be set to a value appropriate to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash operation. 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
B02_RWSC	Bank0+2 Read Wait State Control. This field is used to control the number of wait-states to be added to the flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash operation. Shown below are the maximum operating frequencies for legal APC and RWSC settings based on estimated flash access times at 150 °C. 0 MHz –20 MHz, APC =RWSC=0 > 20 MHz –40 MHz, APC =RWSC=1 > 40 MHz –60 MHz, APC =RWSC=2 > 60 MHz –80 MHz, APC =RWSC=3 > 80 MHz –100 MHz, APC =RWSC=4 >100 MHz –120 MHz, APC =RWSC=5 00000 No additional wait-states are added 00001 1 additional wait-state is added 00010 2 additional wait-states are added ... 111111 31 additional wait-states are added
B02_RWWC	Bank0+2 Read-While-Write Control. This 3-bit field defines the controller response to flash reads while the array is busy with a program (write) or erase operation. 0-- 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.

Table 682. PFCR0 field descriptions (continued)

Field	Description
B02_P1_BCFG	Bank0+2, Port 1 Page Buffer Configuration. This field controls the configuration of the four page buffers in the PFLASH controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. 00 All four buffers are available for any flash access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.
B02_P1_DPFE	Bank0+2, Port 1 Data Prefetch Enable. This field enables or disables prefetching initiated by a data read access. 0 No prefetching is triggered by a data read access 1 If page buffers are enabled (B02_P1_BFE=1), prefetching is triggered by any data read access
B02_P1_IPFE	Bank0+2, Port 1 Instruction Prefetch Enable. This field enables or disables prefetching initiated by an instruction fetch read access. 0 No prefetching is triggered by an instruction fetch read access 1 If page buffers are enabled (B02_P1_BFE=1), prefetching is triggered by any instruction fetch read access
B02_P1_PFLM	Bank0+2, Port 1 Prefetch Limit. This field controls the prefetch algorithm used by the PFLASH controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. 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*.
B02_P1_BFE	Bank0+2, Port 1 Buffer Enable. This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset, enabling the page buffers. 0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

Table 682. PFCR0 field descriptions (continued)

Field	Description
B02_P0_BCFG	Bank0+2, Port 0 Page Buffer Configuration. This field controls the configuration of the four page buffers in the PFLASH controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. 00 All four buffers are available for any flash access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses. This field is set to 2b11 by hardware reset.
B02_P0_DPFE	Bank0+2, Port 0 Data Prefetch Enable. This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access
B02_P0_IPFE	Bank0+2, Port 0 Instruction Prefetch Enable. This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. 0 No prefetching is triggered by an instruction fetch read access 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access
B02_P0_PFLM	Bank0+2, 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*.
B02_P0_BFE	Bank0+2, 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.

35.4.3.2 Platform Flash Configuration Register 1 (PFCR1)

This register defines the configuration associated with flash memory bank1. This typically corresponds to the optional “data flash”. This register should only be written with 32-bit write operations to avoid any issues associated with register "incoherency" caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

Note: This register should not be updated directly from flash memory. Before modifying the wait states:

1. Transfer code execution to RAM.

2. Modify this register.

3. Return to the execution from flash memory.

Offset 0x020 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	B1_APC					0	0	0	0	0	B1_RWSC					B1_RWWC2
W																
Reset	0	1	1	0	1	0	0	0	0	0	0	1	1	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B1_RWWC1	0	0	0	0	0	0	B1_P1_BFE	B1_RWWC0	0	0	0	0	0	0	B1_P0_BFE
W																
Reset	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1

Figure 776. PFLASH Configuration Register 1 (PFCR1)

Table 683. PFCR1 field descriptions

Field	Description
B1_APC	Bank1 Address Pipelining Control. This field is used to control the number of cycles between flash array access requests. This field must be set to a value appropriate to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash operation. 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
B1_RWSC	Bank1 Read Wait State Control. This field is used to control the number of wait-states to be added to the flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH. The required settings are documented in the SoC specification. Higher operating frequencies require non-zero settings for this field for proper flash operation. Shown below are the maximum operating frequencies for legal APC and RWSC settings based on estimated flash access times at 150 °C. 0 MHz –20 MHz, APC =RWSC=2 > 20 MHz –40 MHz, APC =RWSC=4 > 40 MHz –60 MHz, APC =RWSC=7 > 60 MHz –80 MHz, APC =RWSC=9 > 80 MHz –100 MHz, APC =RWSC=11 >100 MHz –120 MHz, APC =RWSC=13 00000 No additional wait-states are added 00001 1 additional wait-state is added 00010 2 additional wait-states are added ... 111111 31 additional wait-states are added
B1_RWWC	Bank1 Read-While-Write Control. This 3-bit field defines the controller response to flash reads while the array is busy with a program (write) or erase operation. 0-- This state should be avoided. 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
B1_P1_BFE	Bank1, Port 1 Buffer Enable. This bit enables or disables read hits from the 64-bit holding register. It is also used to invalidate the contents 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.
B1_P0_BFE	Bank1, Port 0 Buffer Enable. This bit enables or disables read hits from the 64-bit holding register. It is also used to invalidate the contents 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.

35.4.3.3 Platform Flash Access Protection Register (PFAPR)

The PFLASH Access Protection Register (PFAPR) is used to control read and write accesses to the flash based on system master number. Prefetching capabilities are defined on a per master basis. This register also defines the arbitration mode between the 2 AHB ports for the platform flash controller. This register should only be written with 32-bit write operations to avoid any issues associated with register "incoherency" caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

The contents of the register are loaded from location 0x203E00 of the shadow region in the code flash (bank0) array at reset. To temporarily change the values of any of the fields in the PFAPR, a write to the IPS-mapped register is performed. To change the values loaded into the PFAPR *at reset*, the word location at address 0x203E00 of the shadow region in the flash array must be programmed using the normal sequence of operations. The reset value shown in *Figure 777* reflects an erased or unprogrammed value from the shadow region.

Offset 0x024 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	ARBM		M7PFD	M6PFD	M5PFD	M4PFD	M3PFD	M2PFD	M1PFD	M0PFD
W																
Reset	*	*	*	*	*	*	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M7AP		M6AP		M5AP		M4AP		M3AP		M2AP		M1AP		M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 777. PFLASH Access Protection Register (PFAPR)

Table 684. PFLASH Access Protection Register Field Descriptions

Field	Description
0-5	Reserved, should be cleared.
6-7 ARBM	Arbitration Mode. This 2-bit field controls the arbitration for PFLASH controllers supporting 2 AHB ports. The port arbitration mode is used only when accesses from the 2 AHB ports attempt to simultaneously reference the same flash bank. Simultaneous references to different memory banks are processed concurrently. 00 Fixed priority arbitration with AHB p0 > p1 01 Fixed priority arbitration with AHB p1 > p0 1- Round-robin arbitration

Table 684. PFLASH Access Protection Register Field Descriptions (continued)

Field	Description
8-15 MxPFD	Master x Prefetch Disable (x = 0,1,2,...,7). These bits control whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCRn[B02_Px_DPFE, B02_Px_IPFE, Bx_Py_BFE] bits. 0 Prefetching may be triggered by this master 1 No prefetching may be triggered by this master
16-31 MxAP	Master x Access Protection (x = 0,1,2,...,7). These fields control whether read and write accesses to the flash are allowed based on the master number of the initiating module. 00 No accesses may be performed by this master 01 Only read accesses may be performed by this master 10 Only write accesses may be performed by this master 11 Both read and write accesses may be performed by this master

35.4.4 Functional Description

The platform flash controller interfaces between 2 AHB-Lite 2.v6 system bus master ports and three banks of the flash memory arrays.

The platform flash controller generates three sets of interface signals for the flash banks, including read and write enables, the flash array address, write size, and write data as inputs to each flash bank. The platform flash controller captures read data from the flash banks and drives it onto the AHB. Each flash bank includes data storage for fetched pages on a per AHB port basis, either in the form of 4-entry page buffers (banks 0 and 2) or a 1-entry temporary holding register (bank 1). Pages 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.

Multiple prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch.

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms.

Recall the logic of the platform flash controller is structured to support simultaneous AHB accesses from the two ports fully in parallel when the references are targeted to different memory banks. If simultaneous AHB accesses reference the same bank, then arbitration logic within the platform flash controller determines the order the references are granted access to the bank. For more information, see *Section 35.4.4.12: Input Port Arbitration*.

Throughout this discussion, *bkn_* is used as a prefix to refer to the three sets of interface signals, one for the *bk0_*, another for *bk1_* and another for *bk2_*. Also, the nomenclature Bx_Py_RegName is used to reference a program-visible register field associated with bank “x” and port “y”.

35.4.4.1 Basic Interface Protocol

The platform flash controller interfaces to the flash array by driving addresses (*bkn_fl_addr[23:0]*) and read or write enable signals (*bkn_fl_rd_en, bkn_fl_wr_en*).

The read or write enable signal (*bkn_fl_rd_en, bkn_fl_wr_en*) is asserted in conjunction with the reference address for a single rising clock when a new access request is made.

Addresses are driven to the flash array in a flow-through fashion to minimize array access time. When no outstanding access is in progress, the platform flash controller negates *bkn_fl_rd_en* and *bkn_fl_wr_en*. These signals may then change to the next outstanding request in the next cycle.

Accesses are terminated under control of the appropriate read/write wait state control setting. Thus, the access time of the operation is determined by the settings of the wait-state control fields. Access timing can be varied to account for the operating conditions of the SoC (frequency, voltage, temperature) by appropriately setting the fields in the programming model for either bank (b02, b1).

The platform flash controller also has the capability of extending the normal AHB access time by inserting additional wait states for reads and writes. This capability is provided to allow emulation of other memories which have different access time characteristics. The added wait-state specifications are provided by *haddr[28:24]*. These wait-states are applied in addition to the normal wait-states incurred for flash accesses. Refer to *Section 35.4.4.14: Wait-State Emulation* for more details.

Prefetching of next sequential page is blocked when *haddr[28:24]* is non-zero. Buffer hits are also blocked as well, regardless of whether the access corresponds to valid data in one of the page read buffers. These steps are taken to ensure that timing emulation is correct and that excessive prefetching is avoided. In addition, to prevent erroneous operation in certain rare cases, the buffers are invalidated on any non-sequential AHB access with a non-zero value on *haddr[28:24]*.

35.4.4.2 Access Protections

The platform flash controller provides programmable configurable access protections for both read and write cycles from masters via the platform flash controller Access Protection Register (PFAPR). It allows restriction of read and write requests on a per-master basis. This functionality is described in *Section 35.4.3.3: Platform Flash Access Protection Register (PFAPR)*. Detection of a protection violation results in an error response from the platform flash controller on the AHB transfer.

35.4.4.3 Read Cycles - Buffer Miss

Read cycles from the flash array are initiated by driving a valid access address on *bkn_fl_addr[23:0]* and asserting *bkn_fl_rd_en* for the required setup (and hold) time before (and after) the rising edge of *hclk*. The platform flash controller then waits for the programmed number of read wait states before sampling the read data on *bkn_fl_rdata[127:0]*. This data is normally stored in the least-recently updated page read buffer for banks 0 and 2 in parallel with the requested data being forwarded to the AHB. For bank1, the data is captured in the 64-bit temporary holding register as the requested data is forwarded to the AHB bus. Timing diagrams of basic read accesses from the flash array are shown in *Figure 778* through *Figure 783*.

If the flash access was the direct result of an AHB transaction, the page buffer is marked as most-recently-used as it is being loaded. If the flash access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

35.4.4.4 Read Cycles - Buffer Hit

Single cycle read responses to the AHB are possible with the platform flash controller when the requested read access was previously loaded into one of the page buffers associated with banks 0 and 2. 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 64-bit temporary holding registers (one per AHB port) and sequential accesses which “hit” in these registers are also serviced with a zero wait-state response.

35.4.4.5 Write Cycles

In a write cycle, address, write data, and control signals are launched off the same edge of *hclk* at the completion of the first AHB data phase cycle. Write cycles to the flash array are initiated by driving a valid access address on *bkn_fl_addr[23:0]*, driving write data on *bkn_fl_wdata[63:0]*, and asserting *bkn_fl_wr_en*. Again, the controller drives the address and control information for the required setup time before the rising edge of *hclk*, and provides the required amount of hold time. The platform flash controller then waits for the appropriate number of write wait-states before terminating the write operation. On the cycle following the programmed wait state value, the PFLASH2P_LCAasserts *hready_out* to indicate to the AHB port that the cycle has terminated.

35.4.4.6 Error Termination

The platform flash controller follows the standard procedure when an AHB bus cycle is terminated with an ERROR response. First, the platform flash controller asserts *hresp[0]* and negates *hready_out* to signal an error has occurred. On the following clock cycle, the platform flash controller asserts *hready_out* and holds both *hresp[0]* and *hready_out* asserted until *hready_in* is asserted.

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the platform flash controller does not initiate a flash array access.

The second case that can cause an error response to the AHB is when an access is performed to the flash array and is terminated with a flash error response. See *Section 35.4.4.8: Flash Error Response Operation*. This may occur for either a read or a write operation.

The third case that can cause an error response to the AHB is when a write access is attempted to the flash array and is disallowed by the state of the *bkn_fl_ary_access* control input. This case is similar to case 1.

The platform flash controller can also terminate the current AHB access if *hready_in* is asserted before the end of the current bus access. While this circumstance should not occur, this does not result in an error condition being reported, as this behavior is initiated by the AHB master. In this circumstance, the platform flash controller control state machine completes any flash array access in progress (without signaling the AHB) before handling a new access request.

35.4.4.7 Access Pipelining

The platform flash controller does not support access pipelining since this capability is not supported by the flash array. As a result, the APC (Address Pipelining Control) field is

typically set to the same value as the RWSC (Read Wait State Control) field for best performance, that is, Bn_APC = Bn_RWSC. It cannot be less than the RWSC.

35.4.4.8 Flash Error Response Operation

The flash array may signal an error response by asserting *bkn_fl_xfr_err* to terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the platform flash controller does *not* update or validate a bank 0 or 2 page read buffer nor the bank1 temporary holding register. An error response may be signaled on read or write operations. For more information on the specifics related to signaling of errors, including flash ECC, refer to the flash array documentation. For additional information on the system registers which capture the faulting address, attributes, data and ECC information, see the ECSM block guide.

35.4.4.9 Code flash memory bank 0 and 2 page read buffers and prefetch operation

The logic associated with banks 0 and 2 of the platform flash controller contains four page read buffers which are used to hold data read from the flash array. Each buffer stores 4 pages (4 x 128 bit storage) operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

The organization of each page buffer is described below in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags.

```
struct {                          // bx_py_page_buffer
reg addr[23:4];                   // page address
reg valid;                        // valid bit
reg rdata[127:0];                 // page read data
reg xfr_error;                    // transfer error indicator from
                                     flash array
reg multi_ecc_error;              // multi-bit ECC error indicator
                                     from flash array
regsingle_ecc_error;              // single-bit correctable ECC
                                     indicator from flash array
} bx_py_page_buffer[4];
```

Given this definition, the platform flash controller includes four instantiations of the basic 4 x 128 bit page buffer. These are named: b0_p0, b0_p1, b2_p0 and b2_p1.

For the general case, a page buffer is written at the completion of an error-free flash access and the valid bit asserted. Subsequent flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait-states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in *Section 35.4.4.8: Flash Error Response Operation*, a page buffer is *not* marked as valid if the flash array access terminated with any type of transfer error. However, the result is that flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see *Section 35.4.4.10: Buffer Invalidation*.

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters may be enabled or disabled from triggering prefetches, and triggering may be further restricted

based on whether a read access is for instruction or data. A read access to the platform flash controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a flash array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers may be in one of six states, listed here in prioritized order:

1. Invalid - the buffer contains no valid data
2. Used - the buffer contains valid data 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 array, and the array access is still in progress

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement.

Once the candidate page buffer has been selected, the flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, *the recently-used status is not changed*. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash access.

Multiple 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 (Bx_Py_BFE) must be set, the prefetch limit (Bx_Py_PFLM) must be non-zero and either instruction prefetching (Bx_Py_IPFE) or data prefetching (Bx_Py_DPFE) enabled. Recall the prefetch and buffer enables are defined on a per AHB port in the PFCR0 and PFCR1 registers. Refer to *Section 35.4.3.1: Platform Flash Configuration Register 0 (PFCR0)* and *Section 35.4.3.2: Platform Flash Configuration Register 1 (PFCR1)* for a description of these control fields.

35.4.4.9.1 Inst/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 also be set to enable prefetching. Prefetches are never triggered by write cycles.

35.4.4.9.2 Per-Master Prefetch Triggering

Prefetch triggering may be also controlled for individual bus masters. AHB accesses indicate the requesting master via the *hmaster[3:0]* inputs. Refer to PFAPR description for details on these controls.

35.4.4.9.3 Code flash memory buffer allocation

Allocation of the page 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.

35.4.4.10 Buffer Invalidation

The page read buffers may be invalidated under hardware or software control.

Any falling edge transition of the array’s *bkn_fl_done* signal causes the page read buffers to be marked as invalid. This input is negated by the flash array at the beginning of all program/erase operations as well as in certain other cases. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer 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 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 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.

35.4.4.11 Bank1 Temporary Holding Registers

Recall the bank1 logic within the platform flash controller includes two 64-bit data register (one for each AHB port), 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 appropriate temporary data register and subsequent accesses to this page can hit from this register, if it is enabled (B1_Py_BFE).

The organization of the temporary holding register is described below in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with

up to 64 bits of page read data and several error flags and is the same as an individual bank 0 or 2 page buffer.

```
struct {                  //b1_py_page_buffer
   reg addr[23:4];        //page address
   reg valid;             //valid bit
   reg rdata[63:0];       //page read data
   reg xfr_error;         // transfer error indicator from flash array
   reg multi_ecc_error;   // multi-bit ECC error indicator from flash array
   reg single_ecc_error;//single-bit correctable ECC indicator from flash
                          array
} b1_py_page_buffer;
```

Given this definition, the platform flash controller includes two instantiations of this temporary holding register for bank 1. These are named: b1_p0 and b1_p1.

For the general case, a temporary holding register is written at the completion of an error-free flash access and the valid bit asserted. Subsequent flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait-states as the stored read data is routed from the temporary register back to the requesting bus master.

Any 64-bit AHB access request will require two consecutive accesses to the data flash. On a 64-bit AHB read request, the platform flash controller accesses the data flash to obtain the first word (32-bits) and places it in the temporary holding register, aligned accordingly for a 64-bit container. Then the PFLASH2P_LCA accesses the data flash again to obtain the second word. Upon completion of the second flash access, the 32-bit of data on the flash read data bus are combined with the other half of the requested double-word, stored in the temporary holding register, and sent back to the requesting master. The contents of the second flash access are also written to the temporary holding register, aligned accordingly for a 64-bit container. Any subsequent 64- or 32-bit flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait-states as the stored read data is routed from the temporary register back to the requesting bus master.

On a 32-bit read request, the platform flash controller accesses the data flash to obtain the requested word and places it in the temporary holding register, aligned accordingly for a 64-bit container. The other half of the 64-bit holding register is marked invalid. Any subsequent 32-bit flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait-states as the stored read data is routed from the temporary register back to the requesting bus master

The contents of the holding register are invalidated by the falling edge transition of *b1_fl_done* and on any non-sequential access with a non-zero value on *haddr[28:24]* (to support wait-state emulation) in the same manner as the bank0 page buffers. Additionally, the B1_Py_BFE register bit can be cleared by software to invalidate the contents of the holding register.

As noted in *Section 35.4.4.8: Flash Error Response Operation*, the temporary holding register is *not* marked as valid if the flash array access terminated with any type of transfer error. However, the result is that flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the temporary holding register and validated. Accordingly, one special case needing software invalidation relates to holding register “hits” on flash data 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 temporary holding registers need to be invalidated by software after the first notification of the single-bit ECC event*.

Each bank1 temporary holding register effectively operates like a single page buffer.

35.4.4.12 Input Port Arbitration

For maximum system performance, the platform flash controller fully supports concurrent flash accesses from the two AHB input ports when the references are targeted to different flash banks.

In the event that both AHB ports reference the same flash bank, there is arbitration logic in the module to determine the order the references are granted access to the targeted bank. The 2-bit PFAPR[ARBM] field defines the port arbitration mode and this field can define a fixed priority scheme with either p0 > p1 or p1 > p0 or a round-robin mode where the port given priority simply toggles on every simultaneous bank conflict.

35.4.4.13 Read-While-Write Functionality

The platform flash controller supports various programmable responses for read accesses while the flash is busy performing a write (program) or erase operation. For all situations, the platform flash controller uses the state of the flash array's *bkn_fl_done* output to determine if it is busy performing some type of high-voltage operation, namely, if *bkn_fl_done* = 0, the array is busy.

Specifically, there are two 3-bit read-while-write (Bn_RWWC) control register fields which define the platform flash controller's response to these types of access sequences. There are 4 unique responses that are defined by the Bn_RWWC setting: one immediately reports an error on an attempted read and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- Bn_RWWC = 0b111
  - This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform flash controller module simply stalls any read reference until the flash has completed its program/erase operation. If a read access arrives while the array is busy or if a falling-edge on *bkn_fl_done* occurs while a read is still in progress, the AHB data phase is stalled by negating *hready_out* and saving the address and attributes into holding registers. Once the array has completed its program/erase operation, the platform flash controller uses the saved address and attribute information to create a pseudo address phase cycle to "retry" the read reference and sends the registered information to the array as *bkn_fl_rd_en* is asserted. Once the retried address phase is complete, the read is processed normally and once the data is valid, it is forwarded to the AHB bus and *hready_out* negated to terminate the system bus transfer. See Interrupt Controller chapter for details.
- Bn_RWWC = 0b110
  - This setting is similar to the basic stall-while-write capability provided when Bn_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.
- Bn_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 controller's assertion of the *bkn_fl_abort* signal. The *bkn_fl_abort* signal remains asserted until *bkn_fl_done* is driven high. For this setting, there are no notification interrupts generated.

- Bn_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 controller's assertion of the *bkn_fl_abort* signal and an abort notification interrupt generated. There are two abort notification interrupts, one for each bank.

As detailed above, there are a total of 4 interrupt requests associated with the stall-while-write functionality. These interrupt requests are captured as part of ECSM Interrupt Register and logically summed together to form a single request to the interrupt controller.

Table 685. Platform flash controller Stall-While-Write Interrupts

MIR[n]	Interrupt Description
ECSM.MIR[7]	Platform flash bank0 abort notification, MIR[FB0AI]
ECSM.MIR[6]	Platform flash bank0 stall notification, MIR[FB0SI]
ECSM.MIR[5]	Platform flash bank1 abort notification, MIR[FB1AI]
ECSM.MIR[4]	Platform flash bank1 stall notification, MIR[FB1S1]

For example timing diagrams of the stall-while-write and abort-while-write operations, see *Figure 778* and *Figure 783* respectively.

35.4.4.14 Wait-State Emulation

Emulation of other memory array timings are supported by the platform flash controller on read cycles to the flash. This functionality may be useful to maintain the access timing for blocks of memory which were used to overlay flash blocks for the purpose of system calibration or tuning during code development.

The platform flash controller inserts additional wait-states according to the values of *haddr[28:24]*. 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 686 and *Table 687* show the relationship of *haddr[28:24]* to the number of additional primary wait-states. These wait-states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note that the wait-state specification consists of two components: *haddr[28:26]* and *haddr[25:24]* and effectively extends the flash read by (8 * *haddr[25:24]* + *haddr[28:26]*) cycles.

Table 686. Additional Wait-State Encoding

Memory Address haddr[28:26]	Additional wait-states
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 687 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 687. 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

35.4.4.15 Timing Diagrams

Since platform flash controller is typically used in platform configurations with a cacheless core, the operation of the processor accesses to the platform memories, e.g., flash and SRAM, plays a major role in the overall system performance. Given the core/platform pipeline structure, the platform's memory controllers (PFLASH, PRAM) are designed to provide a zero wait-state data phase response to maximize processor performance. The following

diagrams illustrate operation of various cycle types and responses referenced earlier in this chapter including stall-while-read (*Figure 782*) and abort-while-read (*Figure 783*) diagrams.

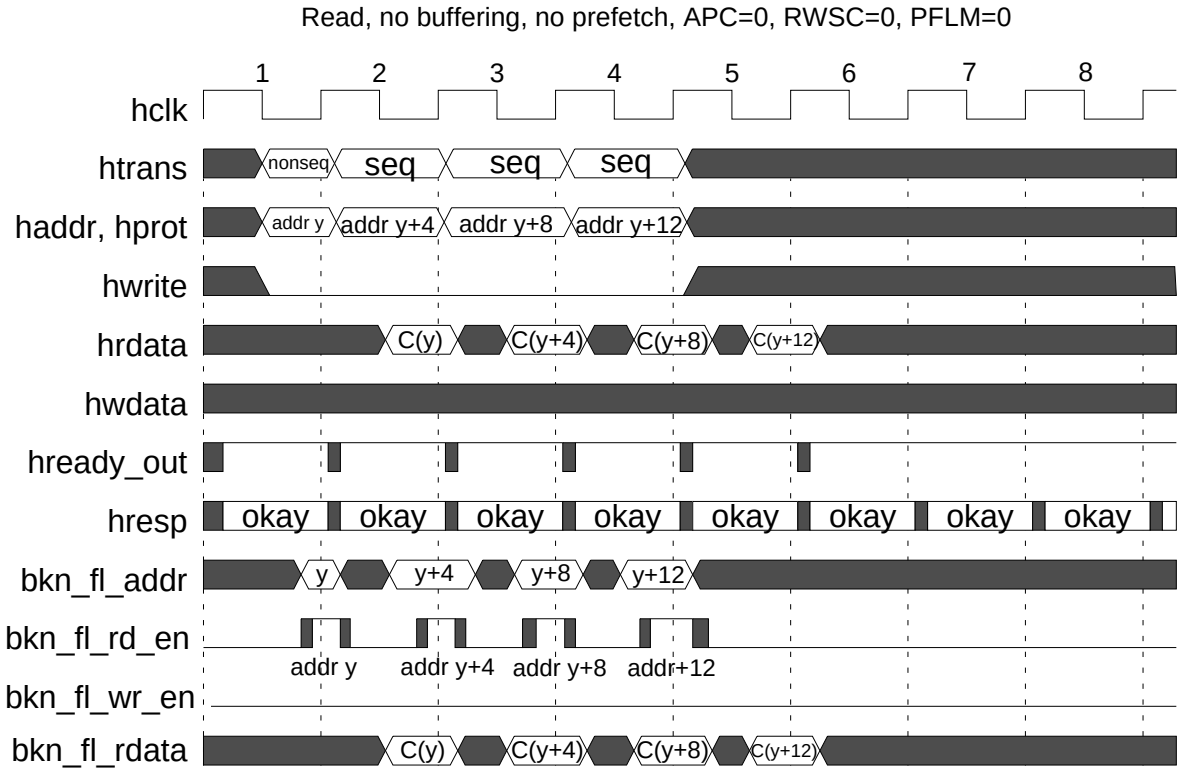


Figure 778. 1-Cycle Access, No Buffering, No Prefetch

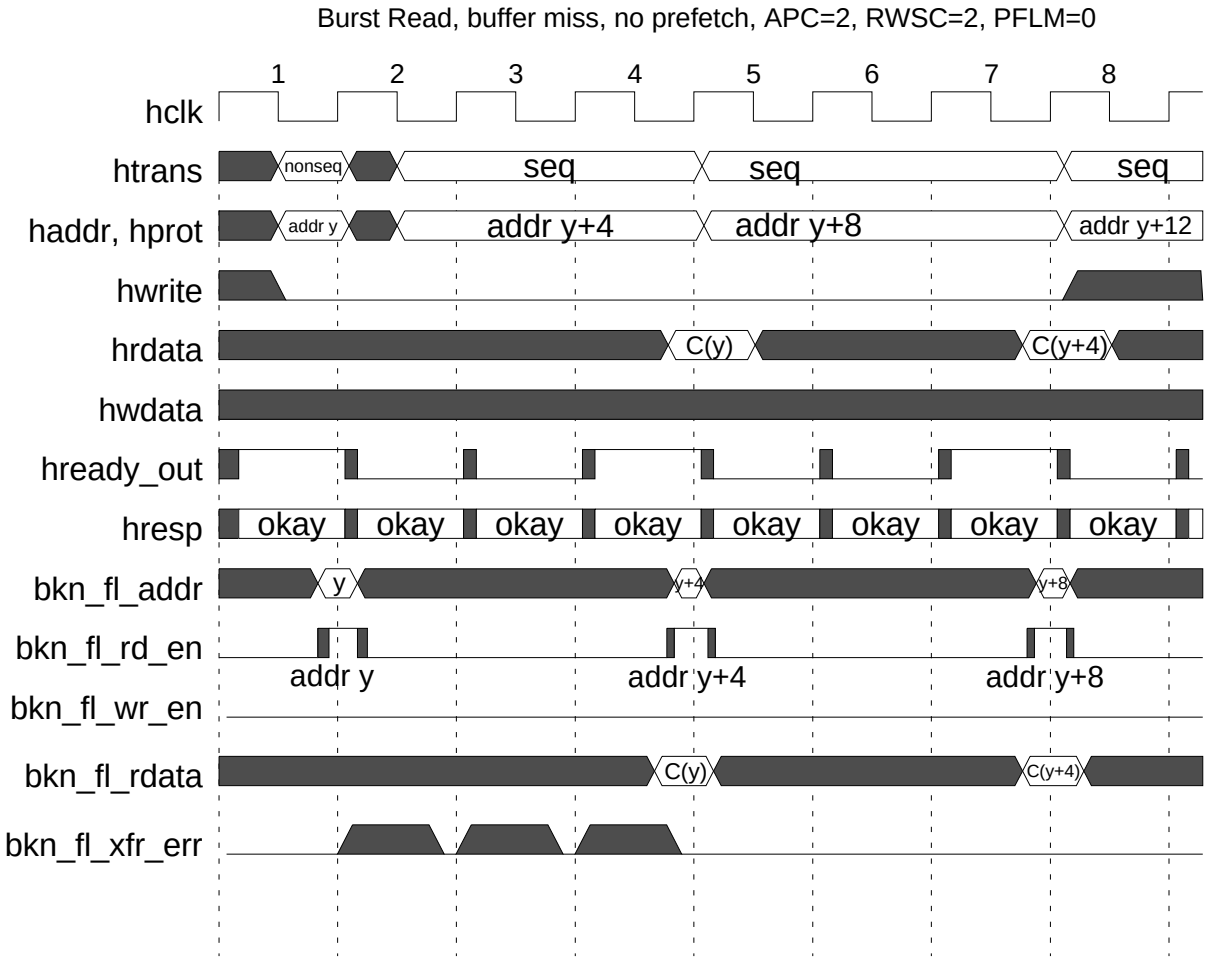


Figure 779. 3-Cycle Access, No Prefetch, Buffering Disabled

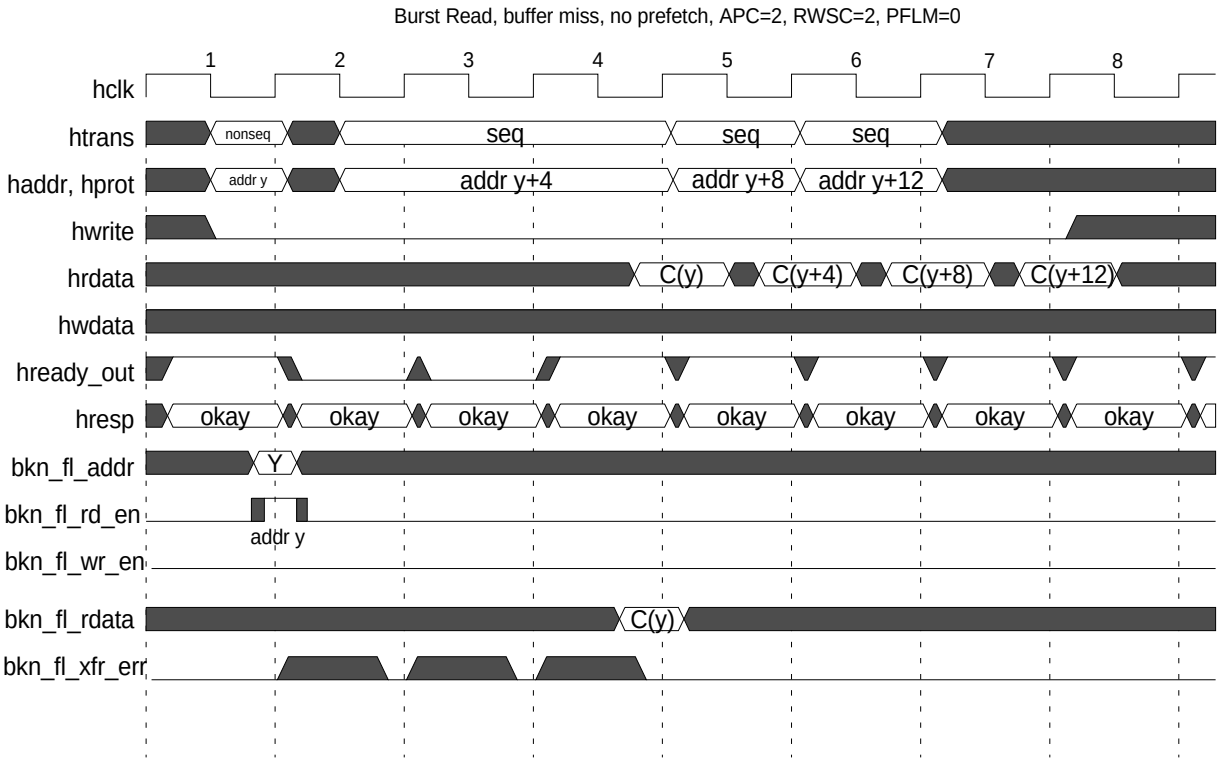


Figure 780. 3-Cycle Access, No Prefetch, Buffering Enabled

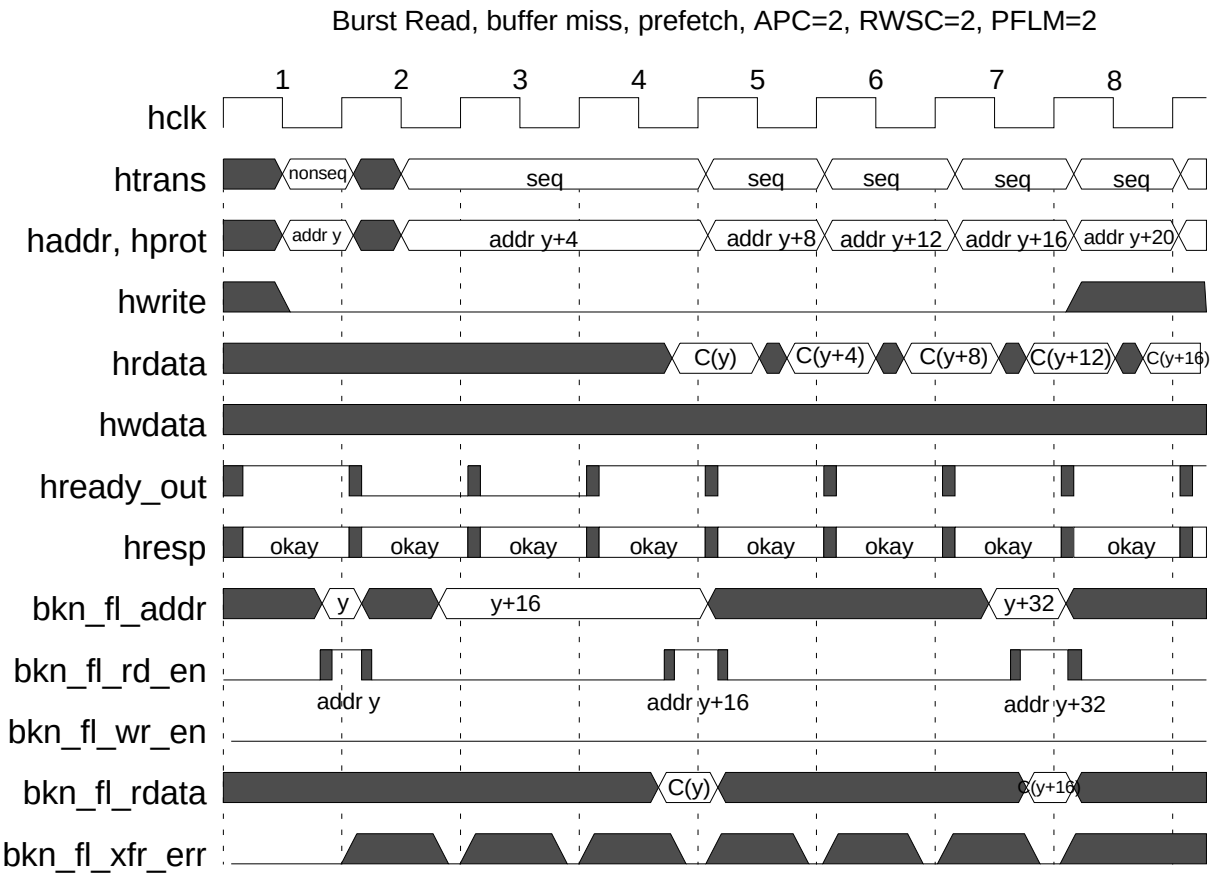


Figure 781. 3-Cycle Access, Prefetch and Buffering Enabled

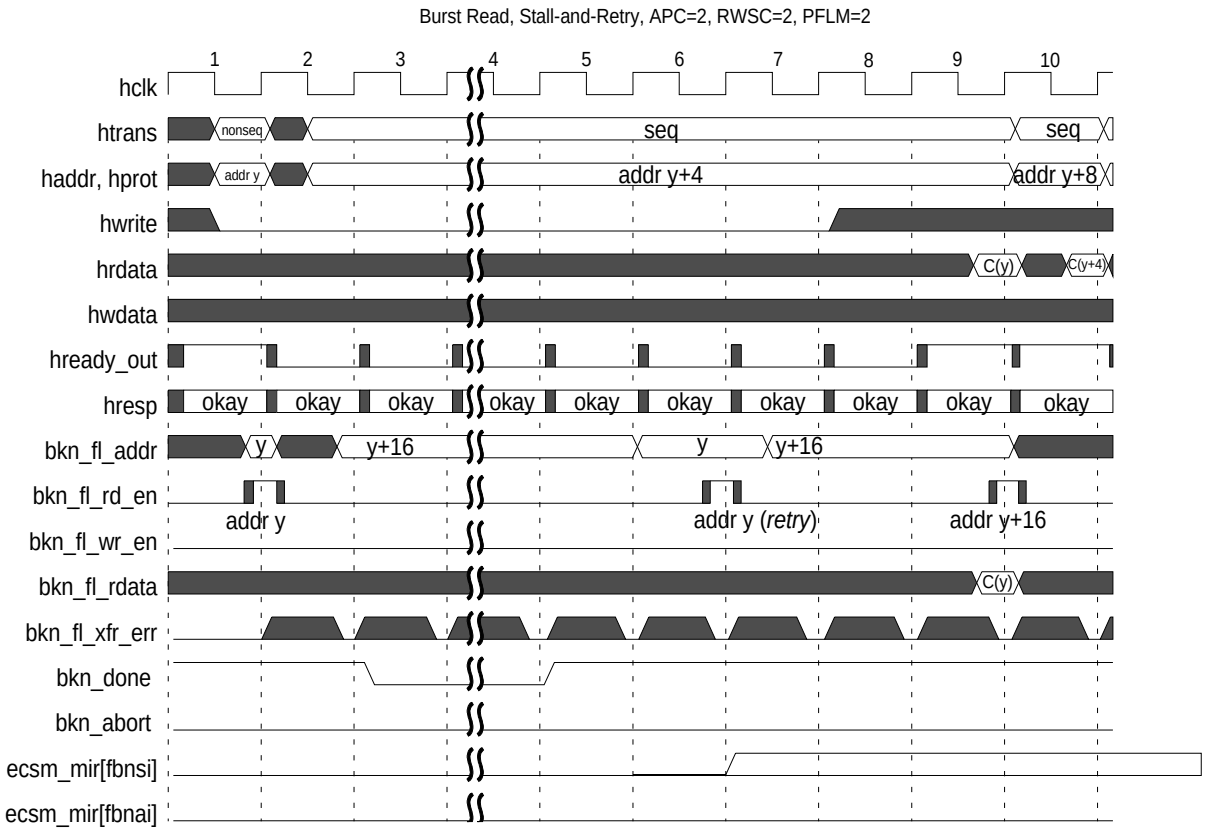


Figure 782. 3-Cycle Access, Stall-and-Retry with Bn_RWWC = 11x

As shown in *Figure 782*, the 3-cycle access to address y is interrupted when an operation causes the *bkn_done* signal to be negated signaling that the array bank is busy with a high-voltage program or erase event. Eventually, this array operation completes (at the end of cycle 4) and *bkn_done* returns to a logical 1. In cycle 6, the platform flash controller module retries the read to address y which was interrupted by the negation of *bkn_done* in cycle 3. Note that throughout cycles 2-9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y+4 in the address phase. Depending on the state of the least-significant-bit of the Bn_RWWC control field, the hardware may also signal a stall notification interrupt (if Bn_RWWC = 110). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FBnSI] (flash bank n stall interrupt).

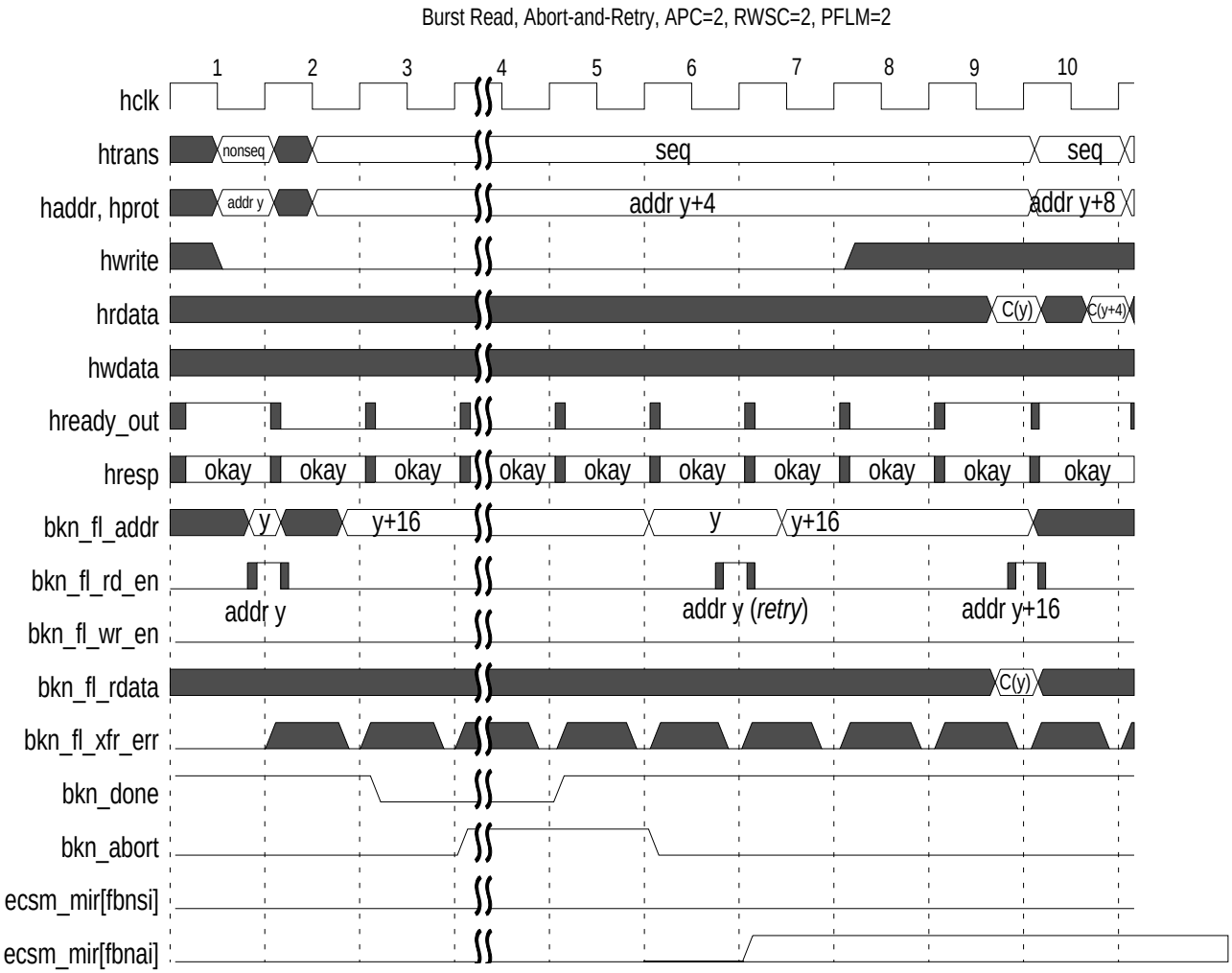


Figure 783. 3-Cycle Access, Abort-and-Retry with Bn_RWWC = 10x

Figure 783 shows the abort-while-write timing diagram. In this example, the 3-cycle access to address y is interrupted when an operation causes the *bkn_done* signal to be negated signaling that the array bank is busy with a high-voltage program or erase event. Based on the setting of Bn_RWWC, once the *bkn_done* signal is detected as negated, the platform flash controller asserts *bkn_abort* which forces the flash array to cancel the high-voltage program or erase event. The array operation completes (at the end of cycle 4) and *bkn_done* returns to a logical 1. It should be noted that the time spent in cycle 4 for *Figure 783* is considerably less than the time in the same cycle in *Figure 782* (because of the abort operation). In cycle 6, the platform flash controller module retries the read to address y which was interrupted by the negation of *bkn_done* in cycle 3. Note that throughout cycles 2-9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y+4 in the address phase. Depending on the state of the least-significant-bit of the Bn_RWWC control field, the hardware may also signal an abort notification interrupt (if Bn_RWWC = 100). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FBnAI] (flash bank n abort interrupt).

36 Register Protection

36.1 Introduction

36.1.1 Overview

The Register Protection Module provides an easy mechanism to protect write access to registers within a module on a per-register basis. The registers that can be protected are typically a subset of the main module registers. *Section 36.6*, details which registers can be protected for each module.

The Register Protection Module sits between the PBRIDGE and the module under protection. This is shown in *Figure 784*.

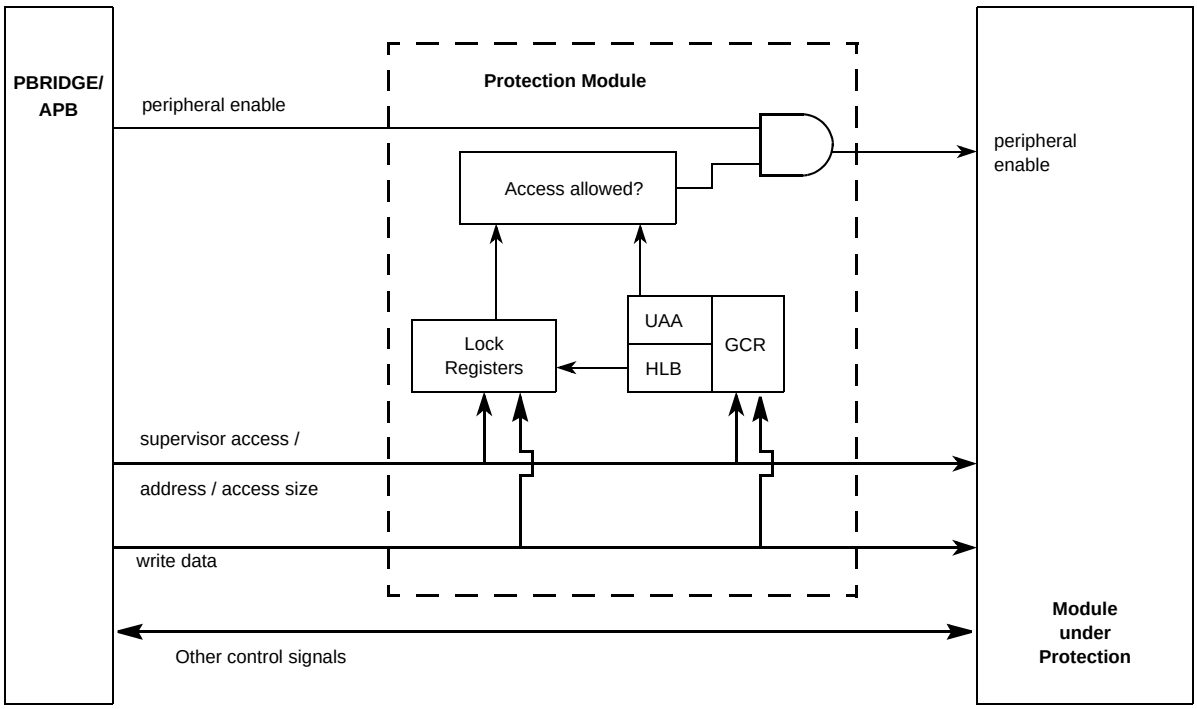


Figure 784. Register Protection Module Block Diagram

Please see *Section 36.6*, for the list of protected registers.

36.1.2 Features

The Register Protection Module includes these distinctive features for each module under protection:

- Restrict access for the module to supervisor mode only.
- Protect a subset of registers located in the first 6 KB of the memory mapped address space
- Provide a mirrored register block at (Base+0x2000) to facilitate automated setting of a register lock bit when the register is written.
- Once register lock bits have been set, they can be protected from changes until reset.

36.1.3 Modes of operation

The Register Protection Module is active when the module under protection is operable.

36.2 External signal description

There are no external signals.

36.3 Memory map and register description

This section provides a detailed description of the memory map of a module using the Register Protection Module. The original 16 KB module memory space is divided into five areas as shown in *Figure 785*.

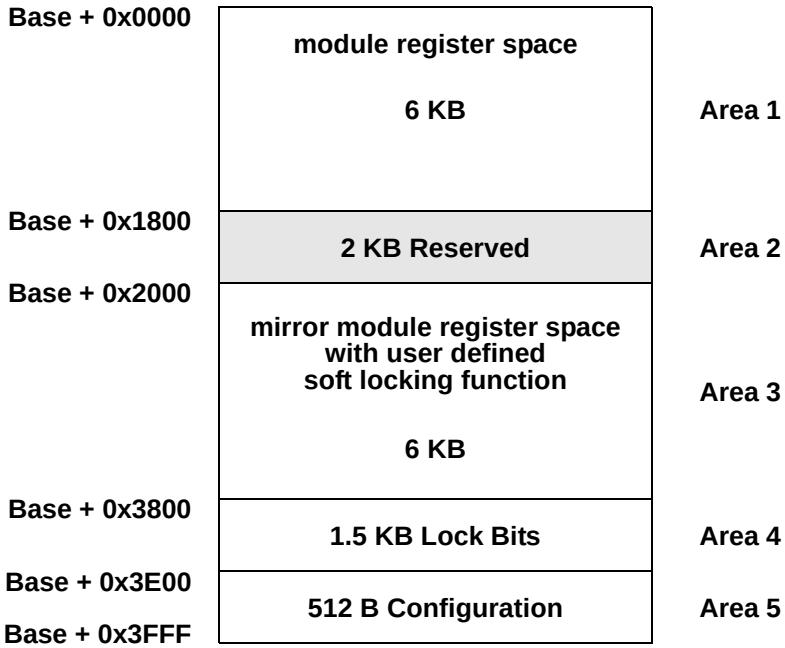


Figure 785. Memory map of module using Register Protection

Area 1 (6 KB) - Registers of module under protection.

Area 2 (2 KB) - Reserved

Area 3 (6 KB) - This is a mirror of the module registers contained in block 1 and is implemented to facilitate setting the register protection lock bits. When a (protectable) register is written in this area, the module register (in area 1) is written as well as updating the corresponding register soft lock bit(s) in Area 4 in the same cycle. For registers that do not have protection available, a write to the register in Area 3 will still perform the register write but will not attempt to update the soft lock bit. Reads to registers in Area 3 are no different to directly reading the register in Area 1.

Area 4 (1.5 KB) - This contains the soft lock bits for each module register that can be protected. There is one soft lock bit per byte of a protected register. 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 (512 bytes) - This area hold 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 within a register.

Accessing unimplemented 32-bit registers in Areas 4 and 5 results in a transfer error.

36.3.1 Memory map

Table 688 gives an overview on the Register Protection Module registers implemented.

Table 688. Register protection memory map

Address offset	Area	Use	Location
0x0000–0x17FF	Area 1	Module register address space	*on page 1257*
0x1800–0x1FFF	Area 2	Reserved	
0x2000–0x37FF	Area 3	Mirror of module register address space with user-defined soft locking function.	*on page 1257*
0x3800	Area 4	Soft Lock Bit Register 0 (SLBR0): soft lock bits 0-3	*on page 1257*
0x3801		Soft Lock Bit Register 1 (SLBR1): soft lock bits 4-7	*on page 1257*
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 1257*
0x3E00–0x3FFB	—	Reserved	
0x3FFC	Area 5	Global Configuration Register (GCR)	*on page 1259*

Note: Reserved registers in area #2 will be handled according to the protected IP (module under protection).

36.3.2 Register description

36.3.2.1 Module register address space (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 SIUL.

36.3.2.2 Module Register and Set Soft Lock Bit (LMR0-6143)

This is memory area 3 which provides mirrored access to the register space MR[0..6143]. Each time a protectable register is written via this mirror, the soft lock bit(s) are updated. Each byte of a register has its own associated soft lock bit as defined in *Section 36.3.2.3*.

36.3.2.3 Soft Lock Bit Register (SLBR0-1535)

These registers contain the soft lock bits for the protectable module registers. This corresponds to area 4 in *Figure 785*.

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

Figure 786. Soft Lock Bit Register (SLBR*n*)

Table 689. SLBR*n* field descriptions

Field	Description
WEn	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 The WE bits allow the user to over-write an existing register that already has soft lock bits set (i.e. one that has previously been written and locked). The mechanism to update a register (that is already locked) is: - Write to the SLBRn register with WEx bits set for whatever SLBn bits are being modified (eg 0xF0 to clear all lock bits) - Re-write the register using the mirrored registers in area 3 which will re-set the locking bits accordingly. If the global protection bit is set, the write enables cannot be enabled and the soft locks remain in force until the next reset. 1 Value is written to SLB 0 SLB is not modified
SLBn	Soft lock bits for one MR*n* 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 MR*n* byte is locked against write accesses 0 Associated MR*n* byte is unprotected and writeable

Table 690 gives some examples how SLBR*n*.SLB and MR*n* go together.

Table 690. 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

Table 690. Soft lock bits vs. protected address (continued)

Soft lock bit	Protected address
SLBR1.SLB2	MR6
SLBR1.SLB3	MR7
SLBR2.SLB0	MR8
...	...

36.3.2.4 Global Configuration Register (GCR)

This register is used to control access to SLB bits related to register protection.

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

Figure 787. Global Configuration Register (GCR)

Table 691. 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 System Integration Unit Lite module.

36.4 Functional description

36.4.1 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)

The specific registers being protected (address and size) depends on the module. This section lists some generic examples to show the mechanism of register protection.

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.

36.4.2 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.

36.4.2.1 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 788 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].

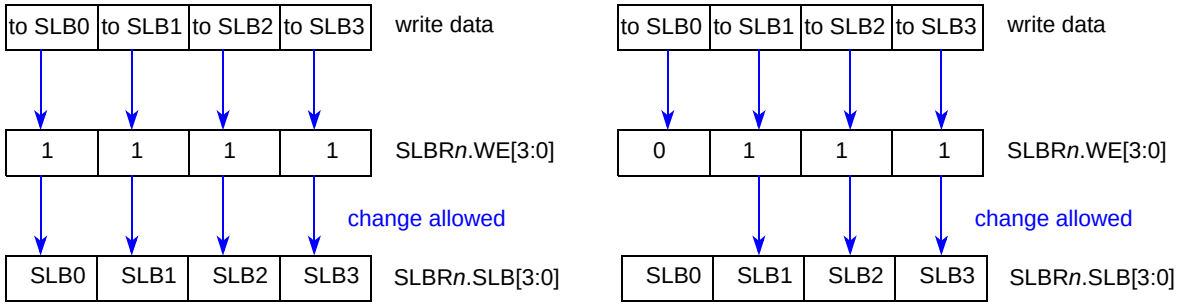


Figure 788. Change Lock Settings Directly Via Area #4

Figure 788 shows four registers that can be protected 8-bit wise. In *Figure 789* registers with 16-bit protection and in *Figure 790* registers with 32-bit protection are shown:

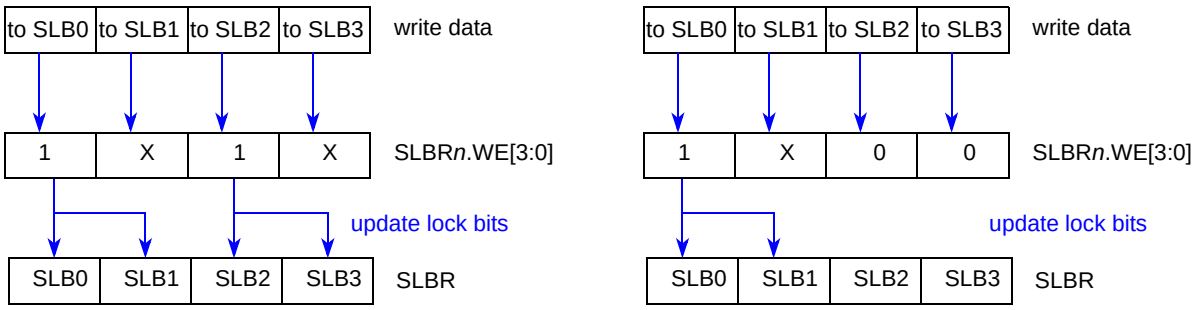


Figure 789. Change Lock Settings for 16-bit Protected Addresses

On the right side of *Figure 789* 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 789* 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 790* 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.

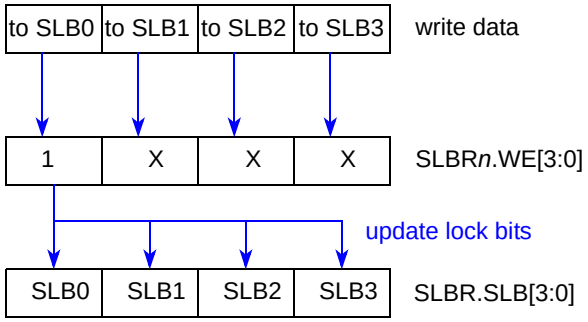


Figure 790. Change Lock Settings for 32-bit Protected Addresses

In *Figure 791* an example is shown which has a mixed protection size configuration:

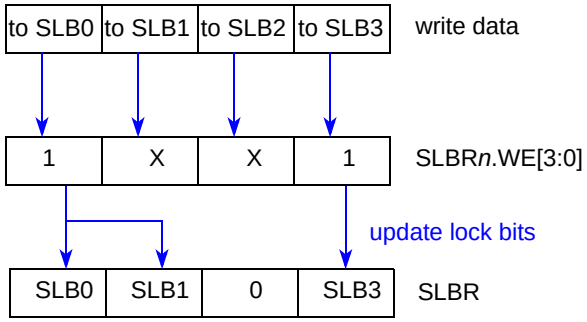


Figure 791. Change Lock Settings for Mixed Protection

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].

36.4.2.2 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 can be used. (Locking can also be enabled directly via the SLRBn registers.) *Figure 792* shows one example:

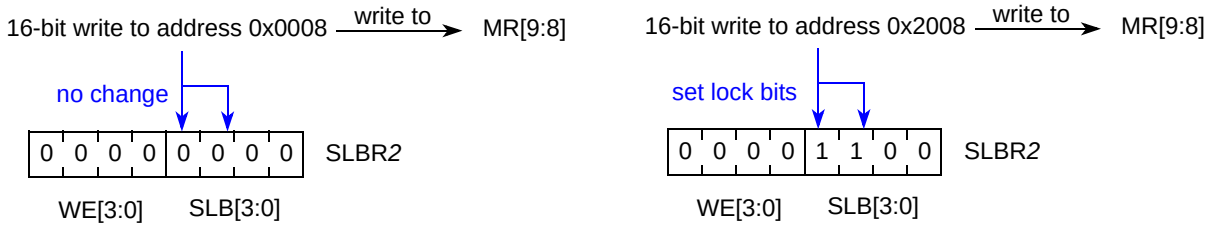


Figure 792. Enable Locking Via Mirror Module Space (Area #3)

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 789*).

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 789*).

Figure 793 shows an example where some addresses are protected and some are not:

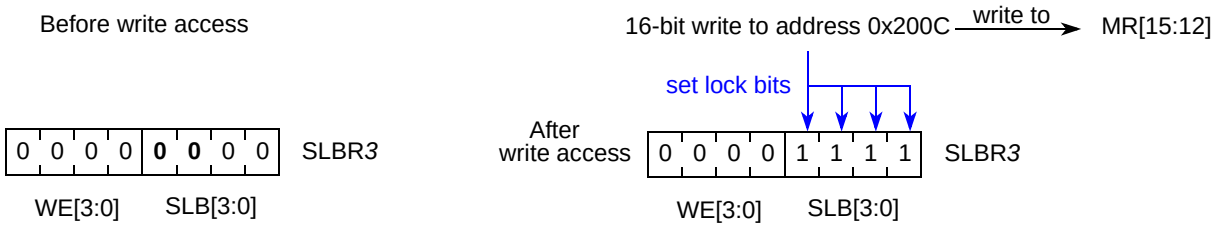


Figure 793. Enable Locking for Protected and Unprotected Addresses

In the example in *Figure 793* addresses 0x0C and 0x0D are unprotected. Therefore their corresponding lock bits SLBR3.SLB[1:0] are always 0b0 (shown in bold). When doing a 16-bit write access to address 0x200C only lock bits SLBR3.SLB[3:2] are set while bits SLBR3.SLB[1:0] stay 0b0. When doing a 32-bit write access to address 0x200C lock bits SLBR3.SLB[3:0] are set

Note: Lock bits are only set automatically during writes to the mirrored address space. The lock bits can also be modified manually.

36.4.2.3 Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in *Section 36.4.2.1: Change lock settings directly via area #4* and *Section 36.4.2.2: 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.

36.4.3 Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to *Figure 785*.

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.

36.5 Reset

The reset state of each individual bit is shown within the Register Description section (See *Section 36.3.2: Register description*). In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in Supervisor Mode only.

36.6 Protected registers

For SPC564Bxx and SPC56ECxx, the Register Protection module is operable on the registers of *Table 692*.

Table 692. Protected registers

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
Code Flash					
Code Flash	MCR	32	C3F88000	000	bits[0:31]
Code Flash	BIU0	32	C3F88000	01C	bits[0:31]
Code Flash	BIU1	32	C3F88000	020	bits[0:31]
Code Flash	BIU2	32	C3F88000	024	bits[0:31]
Data Flash					
Data Flash	MCR	32	C3F8C000	000	bits[0:31]
SIU lite					
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]
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]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
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]
SIUL	PCR46	16	C3F90000	09C	bits[0:15]
SIUL	PCR47	16	C3F90000	09E	bits[0:15]
SIUL	PCR64	16	C3F90000	0C0	bits[0:15]
SIUL	PCR65	16	C3F90000	0C2	bits[0:15]
SIUL	PCR66	16	C3F90000	0C4	bits[0:15]
SIUL	PCR67	16	C3F90000	0C6	bits[0:15]
SIUL	PCR68	16	C3F90000	0C8	bits[0:15]
SIUL	PCR69	16	C3F90000	0CA	bits[0:15]
SIUL	PCR70	16	C3F90000	0CC	bits[0:15]
SIUL	PCR71	16	C3F90000	0CE	bits[0:15]
SIUL	PCR72	16	C3F90000	0D0	bits[0:15]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
SIUL	PCR73	16	C3F90000	0D2	bits[0:15]
SIUL	PCR74	16	C3F90000	0D4	bits[0:15]
SIUL	PCR75	16	C3F90000	0D6	bits[0:15]
SIUL	PCR76	16	C3F90000	0D8	bits[0:15]
SIUL	PCR77	16	C3F90000	0DA	bits[0:15]
SIUL	PCR78	16	C3F90000	0DC	bits[0:15]
SIUL	PCR79	16	C3F90000	0DE	bits[0:15]
SIUL	PCR88	16	C3F90000	0F0	bits[0:15]
SIUL	PCR89	16	C3F90000	0F2	bits[0:15]
SIUL	PCR90	16	C3F90000	0F4	bits[0:15]
SIUL	PCR91	16	C3F90000	0F6	bits[0:15]
SIUL	PCR92	16	C3F90000	0F8	bits[0:15]
SIUL	PCR93	16	C3F90000	0FA	bits[0:15]
SIUL	PCR94	16	C3F90000	0FC	bits[0:15]
SIUL	PCR95	16	C3F90000	0FE	bits[0:15]
SIUL	PCR96	16	C3F90000	100	bits[0:15]
SIUL	PCR97	16	C3F90000	102	bits[0:15]
SIUL	PCR98	16	C3F90000	104	bits[0:15]
SIUL	PCR99	16	C3F90000	106	bits[0:15]
SIUL	PCR100	16	C3F90000	108	bits[0:15]
SIUL	PCR101	16	C3F90000	10A	bits[0:15]
SIUL	PCR102	16	C3F90000	10C	bits[0:15]
SIUL	PCR103	16	C3F90000	10E	bits[0:15]
SIUL	PCR104	16	C3F90000	110	bits[0:15]
SIUL	PCR105	16	C3F90000	112	bits[0:15]
SIUL	PCR106	16	C3F90000	114	bits[0:15]
SIUL	PCR107	16	C3F90000	116	bits[0:15]
SIUL	PCR108	16	C3F90000	118	bits[0:15]
SIUL	PCR109	16	C3F90000	11A	bits[0:15]
SIUL	PCR110	16	C3F90000	11C	bits[0:15]
SIUL	PCR111	16	C3F90000	11E	bits[0:15]
SIUL	PCR112	16	C3F90000	120	bits[0:15]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
SIUL	PCR113	16	C3F90000	122	bits[0:15]
SIUL	PCR114	16	C3F90000	124	bits[0:15]
SIUL	PCR115	16	C3F90000	126	bits[0:15]
SIUL	PCR123	16	C3F90000	136	bits[0:15]
SIUL	PCR124	16	C3F90000	138	bits[0:15]
SIUL	PCR125	16	C3F90000	13A	bits[0:15]
SIUL	PCR126	16	C3F90000	13C	bits[0:15]
SIUL	PCR127	16	C3F90000	13E	bits[0:15]
SIUL	PCR128	16	C3F90000	140	bits[0:15]
SIUL	PCR129	16	C3F90000	142	bits[0:15]
SIUL	PCR130	16	C3F90000	144	bits[0:15]
SIUL	PCR131	16	C3F90000	146	bits[0:15]
SIUL	PCR132	16	C3F90000	148	bits[0:15]
SIUL	PCR133	16	C3F90000	14A	bits[0:15]
SIUL	PCR134	16	C3F90000	14C	bits[0:15]
SIUL	PCR135	16	C3F90000	14E	bits[0:15]
SIUL	PCR136	16	C3F90000	150	bits[0:15]
SIUL	PCR137	16	C3F90000	152	bits[0:15]
SIUL	PCR138	16	C3F90000	154	bits[0:15]
SIUL	PCR139	16	C3F90000	156	bits[0:15]
SIUL	PCR140	16	C3F90000	158	bits[0:15]
SIUL	PCR141	16	C3F90000	15A	bits[0:15]
SIUL	PCR142	16	C3F90000	15C	bits[0:15]
SIUL	PCR143	16	C3F90000	15E	bits[0:15]
SIUL	PCR144	16	C3F90000	160	bits[0:15]
SIUL	PCR145	16	C3F90000	162	bits[0:15]
SIUL	PCR146	16	C3F90000	164	bits[0:15]
SIUL	PCR147	16	C3F90000	166	bits[0:15]
SIUL	PCR148	16	C3F90000	168	bits[0:15]
SIUL	PCR149	16	C3F90000	16A	bits[0:15]
SIUL	PCR150	16	C3F90000	16C	bits[0:15]
SIUL	PCR151	16	C3F90000	16E	bits[0:15]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
SIUL	PCR152	16	C3F90000	170	bits[0:15]
SIUL	PCR153	16	C3F90000	172	bits[0:15]
SIUL	PCR154	16	C3F90000	174	bits[0:15]
SIUL	PCR155	16	C3F90000	176	bits[0:15]
SIUL	PCR156	16	C3F90000	178	bits[0:15]
SIUL	PCR157	16	C3F90000	17A	bits[0:15]
SIUL	PCR158	16	C3F90000	17C	bits[0:15]
SIUL	PCR159	16	C3F90000	17E	bits[0:15]
SIUL	PCR160	16	C3F90000	180	bits[0:15]
SIUL	PCR161	16	C3F90000	182	bits[0:15]
SIUL	PCR162	16	C3F90000	184	bits[0:15]
SIUL	PCR163	16	C3F90000	186	bits[0:15]
SIUL	PCR164	16	C3F90000	188	bits[0:15]
SIUL	PCR165	16	C3F90000	18A	bits[0:15]
SIUL	PCR166	16	C3F90000	18C	bits[0:15]
SIUL	PCR167	16	C3F90000	18E	bits[0:15]
SIUL	PCR168	16	C3F90000	190	bits[0:15]
SIUL	PCR169	16	C3F90000	192	bits[0:15]
SIUL	PCR170	16	C3F90000	194	bits[0:15]
SIUL	PCR171	16	C3F90000	196	bits[0:15]
SIUL	PCR172	16	C3F90000	198	bits[0:15]
SIUL	PCR173	16	C3F90000	19A	bits[0:15]
SIUL	PCR174	16	C3F90000	19C	bits[0:15]
SIUL	PCR175	16	C3F90000	19E	bits[0:15]
SIUL	PCR176	16	C3F90000	1A0	bits[0:15]
SIUL	PCR177	16	C3F90000	1A2	bits[0:15]
SIUL	PCR178	16	C3F90000	1A4	bits[0:15]
SIUL	PCR179	16	C3F90000	1A6	bits[0:15]
SIUL	PCR180	16	C3F90000	1A8	bits[0:15]
SIUL	PCR181	16	C3F90000	1AA	bits[0:15]
SIUL	PCR182	16	C3F90000	1AC	bits[0:15]
SIUL	PCR183	16	C3F90000	1AE	bits[0:15]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
SIUL	PCR184	16	C3F90000	1B0	bits[0:15]
SIUL	PCR185	16	C3F90000	1B2	bits[0:15]
SIUL	PCR186	16	C3F90000	1B4	bits[0:15]
SIUL	PCR187	16	C3F90000	1B6	bits[0:15]
SIUL	PCR188	16	C3F90000	1B8	bits[0:15]
SIUL	PCR189	16	C3F90000	1BA	bits[0:15]
SIUL	PCR190	16	C3F90000	1BC	bits[0:15]
SIUL	PCR191	16	C3F90000	1BE	bits[0:15]
SIUL	PCR192	16	C3F90000	1C0	bits[0:15]
SIUL	PCR193	16	C3F90000	1C2	bits[0:15]
SIUL	PCR194	16	C3F90000	1C4	bits[0:15]
SIUL	PCR195	16	C3F90000	1C6	bits[0:15]
SIUL	PCR196	16	C3F90000	1C8	bits[0:15]
SIUL	PCR197	16	C3F90000	1CA	bits[0:15]
SIUL	PCR198	16	C3F90000	1CC	bits[0:15]
SIUL	PSMI0_3	8	C3F90000	500	bits[0:7]
SIUL	PSMI4_7	8	C3F90000	504	bits[0:7]
SIUL	PSMI8_11	8	C3F90000	508	bits[0:7]
SIUL	PSMI12_15	8	C3F90000	50C	bits[0:7]
SIUL	PSMI16_19	8	C3F90000	510	bits[0:7]
SIUL	PSMI20_23	32	C3F90000	514	bits[0:7]
SIUL	PSMI24_27	32	C3F90000	518	bits[0:7]
SIUL	PSMI28_31	32	C3F90000	51C	bits[0:7]
SIUL	PSMI32_35	32	C3F90000	520	bits[0:7]
SIUL	PSMI36_39	32	C3F90000	524	bits[0:7]
SIUL	PSMI40_43	32	C3F90000	528	bits[0:7]
SIUL	PSMI44_47	32	C3F90000	52C	bits[0:7]
SIUL	PSMI48_51	32	C3F90000	530	bits[0:7]
SIUL	PSMI52_55	32	C3F90000	534	bits[0:7]
SIUL	PSMI56_59	32	C3F90000	538	bits[0:7]
SIUL	PSMI61_63	32	C3F90000	53C	bits[0:7]
SIUL	PSMI64_67	32	C3F90000	540	bits[0:7]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
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	IFMC16	32	C3F90000	1040	bits[0:31]
SIUL	IFMC17	32	C3F90000	1044	bits[0:31]
SIUL	IFMC18	32	C3F90000	1048	bits[0:31]
SIUL	IFMC19	32	C3F90000	104C	bits[0:31]
SIUL	IFMC20	32	C3F90000	1050	bits[0:31]
SIUL	IFMC21	32	C3F90000	1054	bits[0:31]
SIUL	IFMC22	32	C3F90000	1058	bits[0:31]
SIUL	IFMC23	32	C3F90000	105C	bits[0:31]
SIUL	IFCP	32	C3F90000	1080	bits[0:31]
Code Flash					
Code Flash	MCR	32	C3FB0000	032	bits[0:31]
Code Flash	BIU0	32	C3FB0000	01C	bits[0:31]
Code Flash	BIU1	32	C3FB0000	020	bits[0:31]
Code Flash	BIU2	32	C3FB0000	024	bits[0:31]
SSCM					

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
SSCM	DPM_BOOT	32	C3FD8000	018	bits[0:31]
SSCM	DPM_BOOT_KEY	32	C3FD8000	01C	bits[0:31]
MC_ME					
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]
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_STOP0_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]	8	C3FDC000	0C4	bits[0:31]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
MC_ME	ME_PCTL[16..19]	8	C3FDC000	0D0	bits[0:31]
MC_ME	ME_PCTL[20..23]	8	C3FDC000	0D4	bits[0:31]
MC_ME	ME_PCTL[32..35]	8	C3FDC000	0E0	bits[0:31]
MC_ME	ME_PCTL[44..47]	8	C3FDC000	0EC	bits[0:31]
MC_ME	ME_PCTL[48..51]	8	C3FDC000	0F0	bits[0:31]
MC_ME	ME_PCTL[56..59]	8	C3FDC000	0F8	bits[0:31]
MC_ME	ME_PCTL[60..63]	8	C3FDC000	0FC	bits[0:31]
MC_ME	ME_PCTL[68..71]	8	C3FDC000	104	bits[0:31]
MC_ME	ME_PCTL[72..75]	8	C3FDC000	108	bits[0:31]
MC_ME	ME_PCTL[88..91]	8	C3FDC000	118	bits[0:31]
MC_ME	ME_PCTL[92..95]	8	C3FDC000	11C	bits[0:31]
MC_ME	ME_PCTL[104..107]	8	C3FDC000	128	bits[0:7]
MC_MGM					
MC_MGM	CGM_Z0_DCR	8	C3FE0000	0C0	bits[0:7]
MC_MGM	CGM_FEC_DCR	8	C3FE0000	0E0	bits[0:7]
MC_MGM	CGM_FLASH_DCR	8	C3FE0000	120	bits[0:7]
MC_MGM	CGM_OC_EN	8	C3FE0000	373	bits[0:7]
MC_MGM	CGM_OCDS_SC	8	C3FE0000	374	bits[0:7]
MC_MGM	CGM_SC_DC[0..3]	32	C3FE0000	37C	bits[0:31]
MC_MGM	CGM_AC1_SC	8	C3FE0000	388	bits[0:7]
CMU					
CMU	CMU_CSR	32	C3FE0100		bits[24:31]
MC_RGM					
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]
MC_PCU					

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
MC_PCU	PCONF2	32	C3FE8000	008	bits[0:31]
MC_PCU	PCONF3	32	C3FE8000	00C	bits[0:31]
PMC					
PMC	PMC_CFGR	8	C3FE8080	000	bits[0:31]
PMC	PMC_PDMODE	8	C3FE8080	004	bits[0:31]
FLEXRAY					
FLEXRAY	MCR	16	FFFE0000	002	bits[0:15]
FLEXRAY	SYMBADHR	16	FFFE0000	004	bits[0:15]
FLEXRAY	SYMBADLR	16	FFFE0000	006	bits[0:15]
FLEXRAY	STBSCR	16	FFFE0000	008	bits[0:15]
FLEXRAY	MBDSR	16	FFFE0000	00C	bits[0:15]
FLEXRAY	MBSSUTR	16	FFFE0000	00E	bits[0:15]
FLEXRAY	POCR	16	FFFE0000	014	bits[0:15]
FLEXRAY	GIFER	16	FFFE0000	016	bits[0:15]
FLEXRAY	PIER0	16	FFFE0000	01C	bits[0:15]
FLEXRAY	PIER1	16	FFFE0000	01E	bits[0:15]
FLEXRAY	SYMATOR	16	FFFE0000	03E	bits[0:15]
FLEXRAY	SFTOR	16	FFFE0000	042	bits[0:15]
FLEXRAY	SFTCCSR	16	FFFE0000	044	bits[0:15]
FLEXRAY	SFIDRFR	16	FFFE0000	046	bits[0:15]
FLEXRAY	SFIDAFVR	16	FFFE0000	048	bits[0:15]
FLEXRAY	SFIDAFMR	16	FFFE0000	04A	bits[0:15]
FLEXRAY	NMVLR	16	FFFE0000	058	bits[0:15]
FLEXRAY	TICCR	16	FFFE0000	05A	bits[0:15]
FLEXRAY	TI1CYSR	16	FFFE0000	05C	bits[0:15]
FLEXRAY	TI1MTOR	16	FFFE0000	05E	bits[0:15]
FLEXRAY	TI2CR0	16	FFFE0000	060	bits[0:15]
FLEXRAY	TI2CR1	16	FFFE0000	062	bits[0:15]
FLEXRAY	MTSACFR	16	FFFE0000	080	bits[0:15]
FLEXRAY	MTSBCFR	16	FFFE0000	082	bits[0:15]
FLEXRAY	RFRFCTR	16	FFFE0000	09A	bits[0:15]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
FLEXRAY	PCR0–PCR30	16	FFFE0000	00A0–00DC	bits[0:15]
FLEXRAY	MBCCFR0	16	FFFE0000	802	bits[0:15]
FLEXRAY	MBFIDR0	16	FFFE0000	804	bits[0:15]
FLEXRAY	MBCCFR1	16	FFFE0000	80A	bits[0:15]
FLEXRAY	MBFIDR1	16	FFFE0000	80C	bits[0:15]
FLEXRAY	MBCCFR2	16	FFFE0000	810	bits[0:15]
FLEXRAY	MBFIDR2	16	FFFE0000	80C	bits[0:15]
FLEXRAY	MBCCFR127	16	FFFE0000	BFA	bits[0:15]
FLEXRAY	MBFIDR127	16	FFFE0000	BFC	bits[0:15]
FlexCAN 0					
FlexCAN 0	CAN_MCR	32	FFFC0000	000	bits[0:31]
FlexCAN 0	CAN_CTRL	32	FFFC0000	004	bits[0:31]
FlexCAN 0	CAN_RXGMASK	32	FFFC0000	010	bits[0:31]
FlexCAN 0	CAN_RX14MASK	32	FFFC0000	014	bits[0:31]
FlexCAN 0	CAN_RX15MASK	32	FFFC0000	018	bits[0:31]
FlexCAN 0	CAN_IMASK2	32	FFFC0000	024	bits[0:31]
FlexCAN 0	CAN_IMASK1	32	FFFC0000	028	bits[0:31]
FlexCAN 0	CAN_MSGx_CS x = [0:63]	32	FFFC0000	80 + 10.x	bits[0:31]
FlexCAN 0	CAN_MSGx_ID x = [0:63]	32	FFFC0000	84 + 10.x	bits[0:31]
FlexCAN 0	CAN_RXIMRx x = [0:63]	32	FFFC0000	880 + 10.x	bits[0:31]
FlexCAN 1					
FlexCAN 1	CAN_MCR	32	FFFC4000	000	bits[0:31]
FlexCAN 1	CAN_CTRL	32	FFFC4000	004	bits[0:31]
FlexCAN 1	CAN_RXGMASK	32	FFFC4000	010	bits[0:31]
FlexCAN 1	CAN_RX14MASK	32	FFFC4000	014	bits[0:31]
FlexCAN 1	CAN_RX15MASK	32	FFFC4000	018	bits[0:31]

Table 692. Protected registers (continued)

Module	Register	Protected size (bits)	Module base	Register offset	Register size (bits)
FlexCAN 1	CAN_IMASK2	32	FFFC4000	024	bits[0:31]
FlexCAN 1	CAN_IMASK1	32	FFFC4000	028	bits[0:31]
FlexCAN 1	CAN_MSGx_CS x = [0:63]	32	FFFC4000	80 + 10.x	bits[0:31]
FlexCAN 1	CAN_MSGx_ID x = [0:63]	32	FFFC4000	84+ 10.x	bits[0:31]
FlexCAN 1	CAN_RXIMRx x = [0:63]	32	FFFC4000	880+ 10.x	bits[0:31]

37 Software Watchdog Timer (SWT)

37.1 Introduction

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing operation. The servicing operation 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.

37.2 Features

The SWT is clocked by the SIRC.

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

37.3 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.

37.4 External signal description

The SWT module does not have any external interface signals.

37.5 Memory map and register definition

The SWT programming model has seven 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. If the RIA bit in the SWT_CR is set then the SWT generates a system reset on an invalid access otherwise a bus error is generated. If either the HLK or

SLK bits in the SWT_CR are set then the SWT_CR, SWT_TO, SWT_WN, SWT_SK registers are read only.

37.5.1 Memory map

The SWT memory map is shown in *Table 693*. The reset values of SWT_CR, SWT_TO and SWT_WN are device specific. These values are determined by SWT inputs.

Table 693. SWT memory map

Base address: 0xFFF3_8000		
Address offset	**Register**	**Location**
0x0000	SWT Control Register (SWT_CR)	*on page 1277*
0x0004	SWT Interrupt Register (SWT_IR)	*on page 1278*
0x0008	SWT Time-out Register (SWT_TO)	*on page 1279*
0x000C	SWT Window Register (SWT_WN)	*on page 1279*
0x0010	SWT Service Register (SWT_SR)	*on page 1280*
0x0014	SWT Counter Output Register (SWT_CO)	*on page 1280*
0x0018	SWT Service Key Register (SWT_CK)	*on page 1281*
0x001C—0x3FFF	Reserved	

37.5.2 Register descriptions

The following sections detail the individual registers within the SWT programming model.

37.5.2.1 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.

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	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	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0(1)

Figure 794. SWT Control Register (SWT_CR)

1. This bit depends on the WATCHDOG_EN of the *Section 35.2.5.27: Nonvolatile User Options register (NVUSRO)*.

Table 694. SWT_CR Field Descriptions

Field	Description
MAPn	Master Access Protection for Master ID. 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 values are used to service the watchdog
RIA	Reset on Invalid Access. 0 = Invalid access to the SWT generates a bus error 1 = Invalid access to the SWT causes a system reset if WEN=1
WND	Window Mode. 0 = Regular mode, service sequence can be done at any time 1 = Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset. 0 = Generate a reset on a time-out 1 = Generate an interrupt on an initial time-out, reset on a second consecutive time-out
HLK	Hard Lock. This bit is only cleared at reset. 0 = SWT_CR, SWT_TO, SWT_WN and SWT_SK are read/write registers if SLK=0 1 = SWT_CR, SWT_TO, SWT_WN and SWT_SK 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 SWT_WN and SWT_SK are read/write registers if HLK=0 1 = SWT_CR, SWT_TO, SWT_WN and SWT_SK are read only registers
CSL	Default/Stucked = 1. The oscillator clock will always be the 128 kHz. It must be shown as Reserved. Any read-back of this bit returns ‘1’.
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

37.5.2.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

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

Figure 795. SWT Interrupt Register (SWT_IR)

Table 695. 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.

37.5.2.3 SWT Time-Out Register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. This register has read-only access if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Offset 0x008 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	WTO							
W								
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 1	0 0 0 0	0 0 0 0

Figure 796. SWT Time-Out Register (SWT_TO)

Table 696. 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.

37.5.2.4 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.

Offset 0x00C 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	WST																															
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

Figure 797. SWT Window Register (SWT_WN)

Table 697. 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.

37.5.2.5 SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service operation writes used to reset the watchdog timer.

Offset 0x010 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	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																	WSC															
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

Figure 798. SWT Service Register (SWT_SR)

Table 698. 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]). If the SWT_CR[KEY] bit is set, two pseudorandom key values are written to service the watchdog, see *Section 37.6: Functional Description* for details. Otherwise, the sequence 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR[SLK]), the value 0xC520 followed by 0xD928 is written to the WSC field.

37.5.2.6 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.

Offset 0x014 Access: Read Only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	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

Figure 799. SWT Counter Output Register (SWT_CO)

Table 699. SWT_CO Register Field Descriptions

Field	Description
CNT	Watchdog Count. When the watchdog is disabled (SWT_CR[WEN]=0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for 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.

37.5.2.7 SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Offset 0x018 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	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																	SK															
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

Figure 800. SWT Service Register (SWT_SR)

Table 700. SWT_SR Field Descriptions

Field	Description
SK	Service Key.This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SR is (17*SK+3) mod 2^{16}.

37.6 Functional Description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a control register (SWT_CR), an interrupt register (SWT_IR), a time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR), a counter output register (SWT_CO) and a service key register (SWT_SK).

The SWT_CR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT_CR[WEN] bit. The reset value of the SWT_CR[WEN] bit is device specific. If the reset value of this bit is 1, the

watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100 in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_CR[CSL] bit selects which clock (system or oscillator) is used to drive the down counter. The reset value of the SWT_TO register is device specific.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_CR, SWT_TO, SWT_WN and SWT_SK registers are read only. The hard lock is enabled by setting the SWT_CR[HLK] bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_CR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation which consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the time-out period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT_CR[KEY] bit is zero, the fixed sequence 0xA602, 0xB480 is written to the SWT_SR[WSC] field to service the watchdog. If the SWT_CR[KEY] bit is set, then two pseudorandom keys are written to the SWT_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in *Figure 801*. This algorithm will generate a sequence of 2^{16} different key values before repeating. The state of the key generator is held in the SWT_SK register. For example, if SWT_SK[SK] is 0x0100 then the service sequence keys are 0x1103, 0x2136. In this mode, each time a valid key is written to the SWT_SR register, the SWT_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT_SR[WSC] field, SWT_SK[SK] is 0x2136 and the next key sequence is 0x3499, 0x7E2C.

$$SK_{n+1} = (17*SK_n+3) \bmod 2^{16}$$

Figure 801. Pseudorandom Key Generator

Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require up to three system plus seven counter clock cycles.

If window mode is enabled (SWT_CR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT_CR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window

to open due to synchronization logic in the watchdog design. This delay could be up to three system plus four counter clock cycles.

The interrupt then reset bit (SWT_CR[ITR]) controls the action taken when a time-out occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_CR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a one to the SWT_IR[TIF] bit.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for up to six system plus eight counter clock cycles.

The SWT_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT_CR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

38 Error Correction Status Module (ECSM)

38.1 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.

38.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of miscellaneous control functions for the device.

The AIPS is the interface between the Advanced High performance Bus (AHB) interface and on-chip IPS peripherals. IPS peripherals are modules that contain readable/writable control and status registers. The AHB master reads and writes these registers through the AIPS. The AIPS generates module enables, the module address, transfer attributes, byte enables, and write data. These elements then function as inputs to the IPS peripherals.

- IPS — Inter Peripheral Subsytem
- AIPS — interface between the Advanced High performance Bus (AHB) interface and on-chip IPS peripherals
- AHB — Advanced High-performance Bus

38.3 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

38.4 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.

38.4.1 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 701 shows the ECSM's memory map.

Table 701. ECSM memory map

Base address: 0xFFF4_0000		
Address offset	**Register**	**Location**
0x00	Processor Core Type Register (PCT)	*on page 1286*
0x02	SoC-Defined Platform Revision Register (REV)	*on page 1286*
0x04	Reserved	
0x08	IPS On-Platform Module Configuration Register (IMC)	*on page 1286*
0x0C–0x1E	Reserved	
0x1F	Miscellaneous Interrupt Register (MIR)	*on page 1287*
0x20–0x23	Reserved	
0x24	Miscellaneous User-Defined Control Register (MUDCR)	*on page 1288*
0x28–0x42	Reserved	
0x43	ECC Configuration Register (ECR)	*on page 1290*
0x44–0x46	Reserved	
0x47	ECC Status Register (ESR)	*on page 1291*
0x48–0x49	Reserved	
0x4A	ECC Error Generation Register (EEGR)	*on page 1292*
0x4C–0x4F	Reserved	
0x50	Platform Flash ECC Address Register (PFEAR)	*on page 1295*
0x54–0x55	Reserved	
0x56	Platform Flash ECC Master Number Register (PFEMR)	*on page 1295*
0x57	Platform Flash ECC Attributes Register (PFEAT)	*on page 1296*
0x58–0x5B	Reserved	
0x5C	Platform Flash ECC Data Register (PFEDR)	*on page 1296*
0x60	Platform RAM ECC Address Register (PREAR)	*on page 1297*
0x64	Reserved	
0x65	Platform RAM ECC Syndrome Register (PRESR)	*on page 1298*
0x66	Platform RAM ECC Master Number Register (PREMR)	*on page 1300*
0x67	Platform RAM ECC Attributes Register (PREAT)	*on page 1300*
0x68–0x6B	Reserved	
0x6C	Platform RAM ECC Data Register (PREDR)	*on page 1301*

38.4.2 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.

38.4.2.1 Processor Core Type Register (PCT)

The PCT is a 16-bit read-only register specifying the architecture of the processor core in SPC564Bxx and SPC56ECxx 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.

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	1	1	0	0	1	0	0	0	1	0	0	0	1	1	0

Figure 802. Processor Core Type Register (PCT)

Table 702. PCT field descriptions

Field	Description
PCT	Processor Core Type0xE446 identifies the e200z4d core built on Power Architecture technology

38.4.2.2 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.

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

Figure 803. SoC-Defined Platform Revision Register (REV)

Table 703. REV field descriptions

Field	Description
REV	Revision The REV field is specified by an input signal to define a software-visible revision number.

38.4.2.3 IPS On-Platform Module Configuration Register (IMC)

The IMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary 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.

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	1	0	0	1	0	0	1	0	1	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:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 804. IPS On-Platform Module Configuration Register (IMC)

Table 704. IMC 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

38.4.2.4 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 805* and *Table 705*.

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

Figure 805. Miscellaneous Interrupt (MIR) Register

38.4.2.5 Miscellaneous User-Defined Control Register (MUDCR)

Table 705. 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.

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.

Offset: 0x24 Access: Read/write

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

Figure 806. Miscellaneous User-Defined Control (MUDCR) Register

Table 706. MUDCR field descriptions

Field	Description
RAM_WS	Platform RAM wait-state control This bit is used to select whether the platform RAM controller will insert 1-wait state into every read access made to the platform RAM arrays. One wait state is required when the System Frequency is >64MHz 0 The platform RAM controller operates as a 0-wait state controller 1 The platform RAM controller operates as a 1-wait state controller Note: This bit must not be changed while there are ongoing transfer on the SRAM
MUDCR[10:0]	FEC XBAR slave burst enable. MUDCRn enables bursting by the FEC interface to the XBAR slave port controlled by that respective MUDCRn bit. If MUDCRn is asserted, then that XBAR slave port enabled by the bit can accept the bursts allowed by MUDCR8 and MUDCR9. Otherwise, the FEC interface will not burst to the XBAR slave port controlled by that respective MUDCRn bit. Read bursts from that XBAR slave port are enabled by MUDCR8. Write bursts to that XBAR slave port are enabled by MUDCR9. MUDCR0 = Burst enable for haddr[31:29] = 3'h0 MUDCR1 = Burst enable for haddr[31:29] = 3'h1 MUDCR2= Burst enable for haddr[31:29] = 3'h2 MUDCR3= Burst enable for haddr[31:29] = 3'h3 MUDCR4 = Burst enable for haddr[31:29] = 3'h4 MUDCR5 = Burst enable for haddr[31:29] = 3'h5 MUDCR6 = Burst enable for haddr[31:29] = 3'h6 MUDCR7= Burst enable for haddr[31:29] = 3'h7 MUDCR8 = Global Slave Read Burst Enable MUDCR9 = Global Slave Write Burst Enable MUDCR10 = Accumulate Error: 1 = Accumulate 0 = Discard

38.4.2.6 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.

38.4.2.6.1 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.

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

Figure 807. ECC Configuration (ECR) Register

Table 707. ECR field descriptions

Field	Description
ER1BR	Enable SRAM 1-bit Reporting This bit can only be set if the SoC-configurable input enable signal is asserted. 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 This bit can only be set if the SoC-configurable input enable signal is asserted. 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.

38.4.2.6.2 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.

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

Figure 808. ECC Status Register (ESR)

Table 708. 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.

38.4.2.6.3 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.

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

Figure 809. ECC Error Generation Register (EEGR)

Table 709. 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 709. 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 platform RAM controller follows a vector bit ordering scheme where LSB = 0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width For example, consider a 64-bit RAM implementation and ECC organized on a 32-bit boundary.The 32-bit ECC approach requires 7 code bits for each 32-bit word. For a PRAM data width of 64 bits, the actual SRAM is 2x (32b data + 7b for ECC) = 78 bits which is organized as two 39-bit memory banks, "even" bank and "odd" bank. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] of the odd bank is inverted if ERRBIT = 1, then RAM[1] of the odd bank is inverted ... if ERRBIT = 31, then RAM[31] of the odd bank is inverted if ERRBIT = 32, then RAM[0] of the even bank is inverted if ERRBIT = 33, then RAM[1] of the even bank is inverted ... if ERRBIT = 63, then RAM[31] of the even 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 if ERRBIT = 71, then ECC Parity[0] of the even bank is inverted if ERRBIT = 72, then ECC Parity[1] of the even bank is inverted ... if ERRBIT = 77,then ECC Parity[6] of the even bank is inverted For ERRBIT values greater than 77, 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.

38.4.2.6.4 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.

Offset: 0x50 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEAR[31:16]															
W																
Reset:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Figure 810. Platform Flash ECC Address Register (PFEAR)

Table 710. 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.

38.4.2.6.5 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.

Offset: 0x56 Access: Read

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR			
W								
Reset:	0	0	0	0	–	–	–	–

Figure 811. Platform Flash ECC Master Number Register (PFEMR)

Table 711. 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.

38.4.2.6.6 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.

Offset: 0x57 Access: Read

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset:	–	–	–	–	–	–	–	–

Figure 812. Platform Flash ECC Attributes Register (PFEAT)

Table 712. 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]: Cacheable0 = Non-cacheable, 1 = Cacheable Protection[2]: Bufferable0 = Non-bufferable, 1 = Bufferable Protection[1]: Mode0 = User mode, 1 = Supervisor mode Protection[0]: Type0 = I-Fetch, 1 = Data

38.4.2.6.7 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.

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:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 813. Platform Flash ECC Data Register (PFEDR)

Table 713. 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.

38.4.2.6.8 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.

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:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 814. Platform RAM ECC Address Register (PREAR)

Table 714. 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.

38.4.2.6.9 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.

Offset: 0x65 Access: Read

	0	1	2	3	4	5	6	7
R	RESR							
W								
Reset:	–	–	–	–	–	–	–	–

Figure 815. Platform RAM ECC Syndrome Register (PRESR)

Table 715. 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 716* associates the upper 7 bits of the syndrome with the data bit in error.

Table 716 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 716. 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]

Table 716. RAM syndrome mapping for single-bit correctable errors (continued)

PRESR[RESR]	Data bit in error
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]
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]

Table 716. RAM syndrome mapping for single-bit correctable errors (continued)

PRESR[RESR]	Data bit in error
0x4a	DATA ODD BANK[1]
0x4c	DATA ODD BANK[0]
0x03,0x05.....0x4d	Multiple bit error
> 0x4d	Multiple bit error

38.4.2.6.10 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.

Offset: 0x66 Access: Read

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR			
W								
Reset:	0	0	0	0	–	–	–	–

Figure 816. Platform RAM ECC Master Number Register (PREMR)

Table 717. 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.

38.4.2.6.11 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.

Offset: 0x67 Access: Read

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset:	–	–	–	–	–	–	–	–

Figure 817. Platform RAM ECC Attributes Register (PREAT)

Table 718. 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

38.4.2.6.12 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.

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:	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 818. Platform RAM ECC Data Register (PREDR)

Table 719. 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.

38.4.3 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.

39 Self-Test Control Unit (STCU)

39.1 Introduction

The Self-Test Control Unit (STCU) is a component within the overall Safety Integrity Subsystem. The STCU controls the sequencing of the device's self test before the primary user application starts running applications software (Apps SW). The goal of the self test is to detect physical defects in embedded memories with enough coverage to meet the required Safety Integrity Level (SIL) of the system.

To the user, the purpose of the STCU is to display the results of the self test.

39.1.1 Acronyms, abbreviations, and terms

Table 720 contains acronyms, abbreviations, and terms used in this document.

Table 720. Acronyms and abbreviated terms

Term	Meaning
Apps SW	User applications software
BIST	Built-In Self Test
CF	Critical faults
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code (used for internal STCU testing)
HW	Hardware in general
IPS	Integrated Peripheral System Bus Interface
MBIST	Memory Built-In Self Test
MC_RGM	Reset generation module
NCF	Non-critical faults
SIL	Safety Integrity Level (industry standard)
Safety Integrity Subsystem	Collection of hardware and software working together to implement the required SIL
SIR	Stay in reset (type of fault)
SSCM	System Status and Configuration Module
SW	Software in general
Integrity SW	Safety integrity software (a component within the Safety Integrity Subsystem)
WDG	Watchdog Timers

39.2 STCU main features

The STCU cannot be started by software.

The STCU features include the following:

- Performs a one-time self test after a reset trigger event
- Provides register interfaces for both software and hardware:
  - Hardware: SSCM write-one-time register interface
  - Software: IPS read-only register interface
- Manages 40 MBISTs (embedded memory blocks)
- Performs self-checking: The Self Checker monitors critical internal signals during the self test.
- Provides a rich set of status and error information:
  - Timeout flags if the self test does not start or finish within a limited amount of time
  - Status flags for individual MBIST operations
  - Flags for STCU internal errors
- Software can confirm the integrity of the CRC status information by directly comparing the expected and actual results the CRC operations.

39.3 Block diagram and components

Figure 819 shows a block diagram of the STCU.

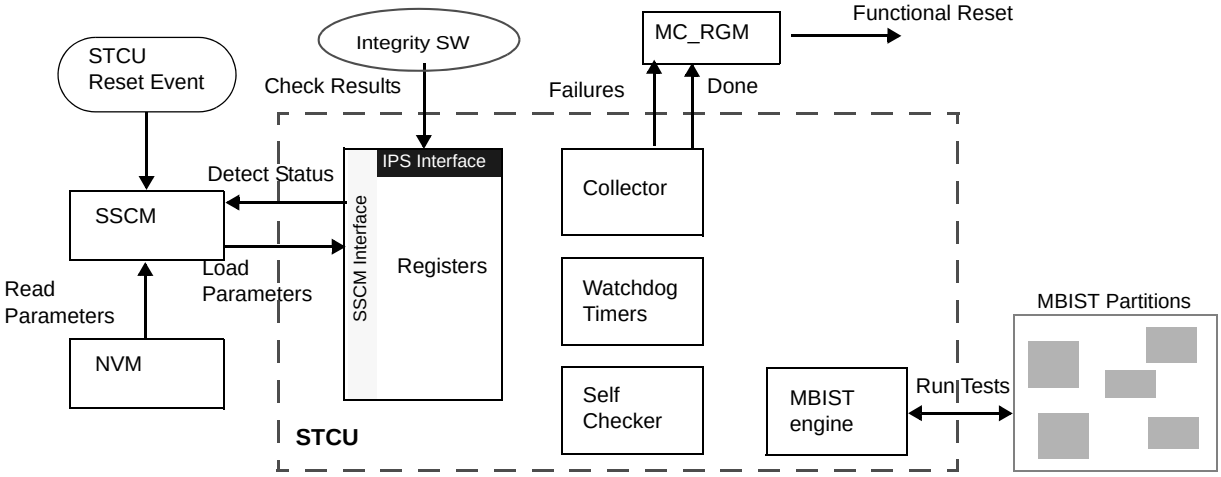


Figure 819. STCU block diagram

The main components of the STCU are:

Registers: Hold the self-test parameters and status flags: scheduling activity, Critical/Non-Critical/Stay-in-Reset fault mapping, and CRC expected values. See *Section 39.5: Memory map and register definition*.

The IPS and SSCM interfaces provide access:

- *SSCM interface*—The SSCM uses this interface to program the STCU's self-test parameters without CPU intervention. A write-once mechanism disables the

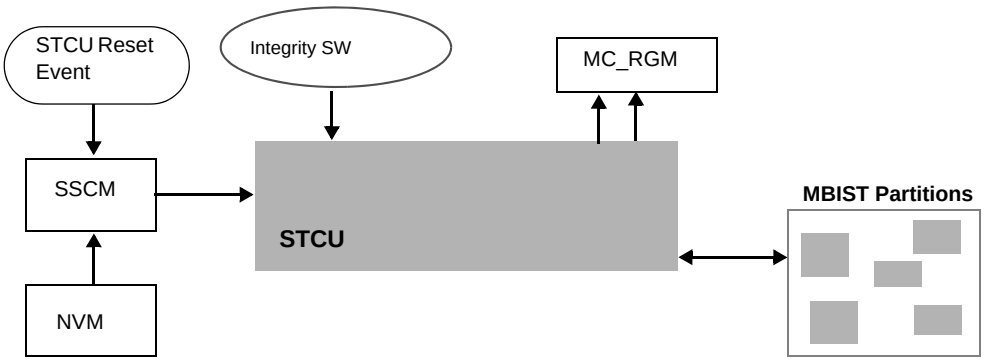
SSCM interface to prevent the STCU parameters from being reloaded after the self test has been performed.

  - *IPS interface*—Software running on the CPUs use this slave bus to access registers.
- ***Self Checker***: Performs a CRC test on a sampled set of selected internal signals when the STCU is running
- ***Collector***: Collects and updates the status and error conditions related to the MBIST execution and STCU internal operation. The Collector sends the BIST results and signals to MC_RGM to begin a functional reset (unless a stay-in-reset fault is encountered).
- ***Watchdog Timers:*** Provide timeout mechanisms to protect against the following:
  - Dead-lock or runaway condition during the self test
  - STCU is activated but the self test is not run
- ***MBIST engine***: Manages the testing of embedded memory blocks.

39.4 The Safety Integrity Subsystem

Figure 820 shows the STCU as a component of the Safety Integrity Subsystem on the device.

Figure 820. The STCU within the Safety Integrity Subsystem



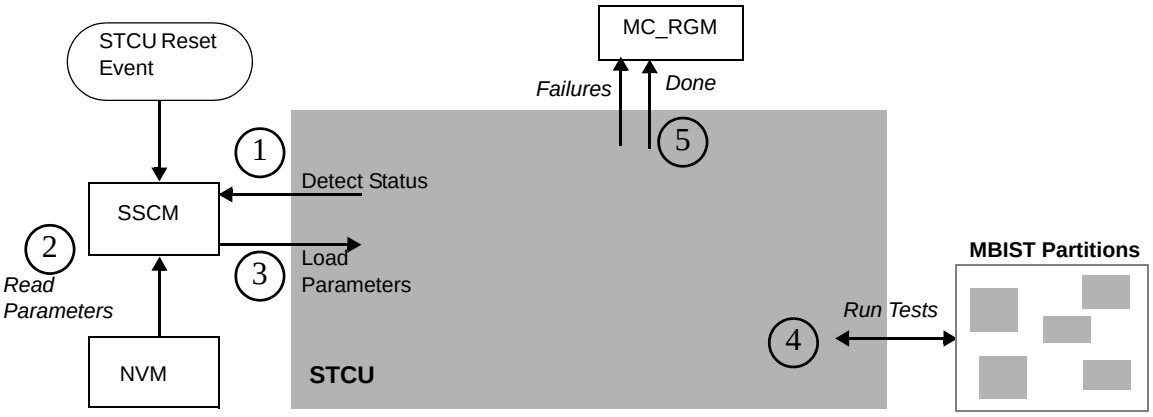
Component (1)	Description
CPU or Integrity SW	The Safety Integrity Software checks the STCU status before passing control over to Integrity SW.
MBIST Partitions	The set of individual embedded memory blocks included in the self test. (See *Table 732*)
MC_RGM	Reset generation module
NVM	The Flash nonvolatile memory contains the initial self-test parameters.
SSCM	The System Status and Configuration Module is the central control for device configuration after reset.
STCU	The Self Test Control Unit manages the device self test.
STCU Reset Event	The following reset events trigger the SSCM to activate the STCU: – Power-on reset – Destructive reset – External reset

1. Components are the hardware and software that make up a subsystem. Events that affect subsystem behavior are also included.

When an STCU Reset Event occurs, the device goes through a two-phase boot sequence:

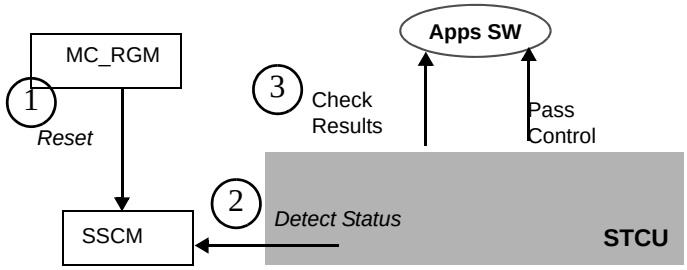
1. Self-test phase: See *Figure 821*.
2. Functional-reset phase: See *Figure 822*.

Figure 821. Boot sequence phase 1: Self test



1. After an STCU Reset Event, the SSCM detects that the device self test has not been run yet.
2. The SSCM reads the self-test parameters from Flash nonvolatile memory (NVM).
3. The SSCM loads the self-test parameters into the STCU and passes control over to the STCU. For details, refer to *Section 39.4.1: Default setup after the boot sequence phase 1*.
4. The STCU manages the MBISTs and updates its internal status.
5. If faults are detected, the STCU reports the test failures to the MC_RGM.
6. The STCU signals the MC_RGM that the tests are complete, and the boot sequence proceeds to the next phase (see *Figure 822*). However, if a SIR fault occurs, the STCU keeps the device in reset until an STCU Reset Event is applied.

Figure 822. Boot sequence phase 2: Functional reset



1. The MC_RGM triggers a functional reset.
2. The SSCM detects that the device self test has been run and passes control over to the CPUs.
3. The integrity SW checks the results of the self test.
4. If the integrity SW check passes, the device can be considered as OK. If there is any fault, it takes an appropriate action.

39.4.1 Default setup after the boot sequence phase 1

Table 721 shows the STCU registers configuration after the boot sequence phase 1, (parameters loaded from Flash into the STCU).

Table 721. STCU registers configuration after the boot sequence phase 1

STCU register		Note
Name	Value	
STCU_RUN	0x0000_0001	MBIST test activation
STCU_CFG	0x1000_0000	The first MBIST scheduled is cluster #0 STCU clock is the system clock: FIRC source
STCU_WDGG	0x0000_0003	The watchdog 10 bit counter granularity is 2^{11b} x 1024 STCU clock cycles
STCU_CRCE	0x5ecb_2787	This value is to be defined and programmed in shadow Flash by the user.
STCU_CRCR	don't care	This value will be updated by the STC after the MBISR run.
STCU_ERR	0x0000_0000	– WDG timeout issues a SIR fault – CRC mismatch issues a NCI fault – Engine error issues a NCI fault – Invalid pointer issues a NCI fault
STCU_MBSL	0x0000_0000	—
STCU_MBSH	0x0000_0000	—
STCU_MBEL	0x0000_0000	—
STCU_MBEH	0x0000_0000	—
STCU_MBCFML	0x0000_0000	Non critical fault configured for every memory cluster
STCU_MBCFMH	0x0000_0000	
STCU_MBSFML	0x0000_0000	No stay in the Reset fault configured for every memory cluster
STCU_MBSFMH	0x0000_0000	
STCU_MB_CTRL0	0x9103_0000	– Concurrent mode selected – Next MIST pointer is 0x11 – MBIST0 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL1	0x9203_0000	– Concurrent mode selected – Next MIST pointer is 0x12 – MBIST1 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL2	00x9303_0000	– Concurrent mode selected – Next MIST pointer is 0x13 – MBIST2 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL3	0x9403_0000	– Concurrent mode selected – Next MIST pointer is 0x14 – MBIST3 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL4	0x9503_0000	– Concurrent mode selected – Next MIST pointer is 0x15 – MBIST4 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL5	0x9603_0000	– Concurrent mode selected – Next MIST pointer is 0x16 – MBIST5 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles

Table 721. STCU registers configuration after the boot sequence phase 1

STCU register		Note
STCU_MB_CTRL6	0x9703_0000	– Concurrent mode selected – Next MIST pointer is 0x17 – MBIST7 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL7	0x9803_0000	– Concurrent mode selected – Next MIST pointer is 0x18 – MBIST7 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL8	0x9903_0000	– Concurrent mode selected – Next MIST pointer is 0x19 – MBIST8 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL9	0x9A03_0000	– Concurrent mode selected – Next MIST pointer is 0x1A – MBIST9 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL10	0x9B03_0000	– Concurrent mode selected – Next MIST pointer is 0x1B – MBIST10 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL11	0x9C03_0000	– Concurrent mode selected – Next MIST pointer is 0x1C – MBIST11 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL12	0x9D03_0000	– Concurrent mode selected – Next MIST pointer is 0x1D – MBIST12 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL13	0x9E03_0000	– Concurrent mode selected – Next MIST pointer is 0x1E – MBIST13 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL14	0x9F03_0000	– Concurrent mode selected – Next MIST pointer is 0x1F – MBIST14 run time budget is0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL15	0xA003_0000	– Concurrent mode selected – Next MIST pointer is 0x20 – MBIST15 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL16	0xA103_0000	– Concurrent mode selected – Next MIST pointer is 0x21 – MBIST17 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL17	0xA203_0000	– Concurrent mode selected – Next MIST pointer is 0x22 – MBIST17 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL18	0xA303_0000	– Concurrent mode selected – Next MIST pointer is 0x23 – MBIST18 run time budget is 0x03 x 2^{11b}x 1024 STCU clock cycles

Table 721. STCU registers configuration after the boot sequence phase 1

STCU register		Note
STCU_MB_CTRL19	0xA403_0000	– Concurrent mode selected – Next MIST pointer is 0x24 – MBIST19 run time budget is 0x03 x 2^{11b}x 1024 STCU clock cycles
STCU_MB_CTRL20	0xA503_0000	– Concurrent mode selected – Next MIST pointer is 0x25 – MBIST20 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL21	0xA603_0000	– Concurrent mode selected – Next MIST pointer is 0x26 – MBIST21 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL22	0xA703_0000	– Concurrent mode selected – Next MIST pointer is 0x27 – MBIST22 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL23	0xA803_0000	– Concurrent mode selected – Next MIST pointer is 0x28 – MBIST23 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL24	0XA91C_0000	– Concurrent mode selected – Next MIST pointer is 0x29 – MBIST24 run time budget is 0x1C x 2^{11b}x 1024 STCU clock cycles
STCU_MB_CTRL25	0xAA18_0000	– Concurrent mode selected – Next MIST pointer is 0x2A – MBIST25 run time budget is 0x18 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL26	0xAB18_0000	– Concurrent mode selected – Next MIST pointer is 0x2B – MBIST27 run time budget is 0x18 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL27	0xAC18_0000	– Concurrent mode selected – Next MIST pointer is 0x2C – MBIST27 run time budget is 0x18 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL28	0xAD03_0000	– Concurrent mode selected – Next MIST pointer is 0x2D – MBIST28 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL29	0xAE07_0000	– Concurrent mode selected – Next MIST pointer is 0x2E – MBIST29 run time budget is 0x07 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL30	0xAF2F_0000	– Concurrent mode selected – Next MIST pointer is 0x2F – MBIST30 run time budget is 0x2F x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL31	0xB030_0000	– Concurrent mode selected – Next MIST pointer is 0x30 – MBIST31 run time budget is 0x30 x 2^{11b} x 1024 STCU clock cycles

Table 721. STCU registers configuration after the boot sequence phase 1

STCU register		Note
STCU_MB_CTRL32	0xB12F_0000	– Concurrent mode selected – Next MIST pointer is 0x31 – MBIST32 run time budget is 0x2F x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL33	0xB22F_0000	– Concurrent mode selected – Next MIST pointer is 0x32 – MBIST33 run time budget is 0x2F x 2^{11b}x 1024 STCU clock cycles
STCU_MB_CTRL34	0xB32F_0000	– Concurrent mode selected – Next MIST pointer is 0x33 – MBIST34 run time budget is 0x2F x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL35	0xB430_0000	– Concurrent mode selected – Next MIST pointer is 0x34 – MBIST35 run time budget is 0x30 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL36	0xB530_0000	– Concurrent mode selected – Next MIST pointer is 0x35 – MBIST37 run time budget is 0x30 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL37	0xB630_0000	– Concurrent mode selected – Next MIST pointer is 0x36 – MBIST37 run time budget is 0x30 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL38	0xB703_0000	– Concurrent mode selected – Next MIST pointer is 0x37 – MBIST38 run time budget is 0x03 x 2^{11b} x 1024 STCU clock cycles
STCU_MB_CTRL39	0x7F13_0000	– Concurrent mode selected – Next MIST pointer is 0x10 – MBIST39 run time budget is 0x13 x 2^{11b} x 1024 STCU clock cycles

39.4.2 Changing the default fault grading

The fault mapping in terms of critical fault (CF), non-critical fault (NCF) and stay in reset (SIR) can be configured by modifying parameters in shadow Flash.

The following parameters need to be changed to suit the specific application:

- STCU_MBCFML:STCU_MBCFMH
- STCU_MBSFML:STCU_MBSFMH
- STCU_ERR

The process for programing these parameters is documented in the Flash memory chapter, *Table 658*.

The STCU_CRCE needs to be calculated for the updated parameters. However, to guarantee maximum coverage during self test, the CRC value cannot be calculated offline and the STUC itself needs to be used to calculate the value.

To calculate CRC value, follow the steps provided below:

1. Modify the Self-test parameters (CF, NCF, SIR values).
2. Run the Self-Test.
3. Read the CRCR.
4. Use this CRCR value to program the Final Self-Test parameter (CRCE).

39.4.3 Integrity SW operations

The Integrity SW performs operations, based on the STCU status conditions after the self test. If no error is reported, the Integrity software confirms that the expected and actual values within the CRC registers do not indicate an error. This software confirmation prevents a fault within the STCU itself incorrectly indicating that the self test passed.

39.4.3.1 IReported errors

In the case of reported errors, the Integrity SW should:

- Read the STCU_MBSL flag register to determine which MBISTs failed.
- Read the STCU_MBEL flag register to determine which MBISTs did not finish.
- Read the STCU_ERR register to check whether there has been an internal STCU failure.

39.4.3.2 No reported errors

In the case of no reported errors, the Integrity SW should confirm the following:

- The internal CRC computation result matches the expected value:
  Read the CRCE and CRCR registers to check the coherency with the STCU_ERR[CRCS] flag.

39.5 Memory map and register definition

All registers shown in this section are defined as visible by the IPS interface.

39.5.1 Memory map

The STCU memory map is listed in *Table 722*.

Table 722. STCU register map

Base address: 0xC3FF_4000		
Address offset	**Register**	**Location**
0x0000–0x0004	Reserved	
0x0008	STCU SK Code Register (STCU_SKC)	*on page 1314*
0x000C	STCU Configuration Register (STCU_CFG)	*on page 1314*
0x0010	STCU Watchdog Register Granularity (STCU_WDGG)	*on page 1315*
0x0014	STCU CRC Expected Status Register (STCU_CRCE)	*on page 1316*
0x0018	STCU CRC Read Status Register (STCU_CRCR)	*on page 1317*
0x001C	STCU Error Register (STCU_ERR)	*on page 1317*

Table 722. STCU register map (continued)

Base address: 0xC3FF_4000		
Address offset	**Register**	**Location**
0x0020	STCU Error Key Register (STCU_ERRK)	*on page 1317*
0x0024–0x0038	Reserved	
0x003C	STCU MBIST Status Low Register (STCU_MBSL)	*on page 1319*
0x0040	STCU MBIST Status High Register (STCU_MBSH)	*on page 1321*
0x0044	STCU MBIST End Flag Low Register (STCU_MBEL)	*on page 1324*
0x0048	STCU MBIST End Flag High Register (STCU_MBEH)	*on page 1324*
0x004C	STCU MBIST Status End Key Register (STCU_MBSEK)	*on page 1324*
0x0050	STCU MBIST Critical FM Low Register (STCU_MBCFML)	*on page 1326*
0x0054	STCU MBIST Critical FM High Register (STCU_MBCFMH)	*on page 1326*
0x0058	STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML)	*on page 1327*
0x005C	STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)	*on page 1328*
0x0060	STCU MBIST FM Key Register (STCU_MBFMK)	*on page 1328*
0x0064–0x002FF	Reserved	
(0x0300 + (($k^{(1)}$-1) × 0x4))	STCU MBIST Control Register (STCU_MB_CTRL)	*on page 1330*
0x0780–0x7FFF	Reserved	

1. Here 'k' is a variable representing the repeated register blocks of the multiple MBISTs: k ranges from 1 up to 40.

39.5.2 Register conventions

The following bus operations (contiguous byte enables) are supported:

- Word (32 bits) data read operations
- Low and high halfwords (16 bits, data[31:16] or data[15:0]) data read operations
- Byte (8 bits, data[31:24] or data[23:16] or data[15:8] or data[7:0]) data read operations

The STCU generates a bus transfer error in the following cases:

- A read access to the register addresses not mapped on the peripheral but included in the address space of the peripheral
- A read operation different from byte/halfword/word (free byte enables or other operations) on each register

The registers of the STCU are accessible (read only) in each access mode: user or supervisor.

39.5.3 Detailed register descriptions

39.5.3.1 STCU SK Code Register (STCU_SKC)

The STCU_SKC register implements the security key code mechanism needed to access the write mode to other STCU registers. To unlock the STCU access after

- The Power-On, Destructive or External Reset
- The completion of the STCU run

the SW (IPs bus) or the SSCM interfaces have to apply the following sequence:

1. Write the key1 into the STCU_SKC register
2. Write the key2 into the STCU_SKC register

After the Self-Test sequence has been completed or the Bypass feature has been enabled (forcing the signal tcu_bypass_slftst), the SSCM interface is no more available.

In case of invalid access or sequence (Key1/2 have to be applied consecutively), a transfer error on the IPS or SSCM bus is asserted depending on the selected source. The STCU write access is locked and to unlock the access the sequence has to be applied again.

In case the STCU register access last more cycles than the one defined into the Hard-coded WDG time-out, the STCU write access is locked and the WDG and Register ITF clocks are switched off. Also, in this case, to enable again the write access to the STCU and the WDG and Register ITF clocks, it is required to apply again the sequence. The STCU_SKC register is not readable. The value 00000000h is always returned in case of read operation.

Offset: 0x0008 Access: User write-only

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

Figure 823. STCU SK Code Register (STCU_SKC)

Table 723. STCU_SKC field descriptions

Field	Description
SKC	STCU security key code = ABFC1893h: Key1 to unlock the write access the STCU (when not protected) = 319A6C2Fh: Key2 to unlock the write access the STCU (when not protected)

39.5.3.2 STCU Configuration Register (STCU_CFG)

The STCU_CFG register includes the global configuration of the STCU.

Offset: 0x000C Access: User read/write

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

Figure 824. STCU Configuration Register (STCU_CFG)

Table 724. STCU_CFG field descriptions

Field	Description
PTR	MBIST pointer PTR defines the logical pointer to the first MBIST to be scheduled. 10h to (10h + 34): pointer to MBIST 7Fh: pointer to NIL. No BIST execution.
CLK_CFG	Logic, Memory BIST and STCU core CLK *Clock configuration* CLK_CFG defines the ratio between the sys_clk and the clock used to program the MBIST and STCU core clock. The allowed configurations are the following: 0000: sys_clk 0001: sys_clk/2 0010: sys_clk/3 .... 1101: sys_clk/14 1110: sys_clk/15 1111: sys_clk/16

39.5.3.3 STCU Watchdog Register Granularity (STCU_WDGG)

The STCU_WDGG register defines the granularity of the MBIST watchdog timer that provides protection against dead-lock or runaway conditions during the self test.

When the self test is not run but the STCU is still activated, the bits 15..0 define the timeout before the STCU_ERR[WDTO] bit is set and the STCU core clock is switched off.

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	0	0	0	0	0	0	0	0	0	0	0	0	GMBIST		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Figure 825. STCU Watchdog Register Granularity (STCU_WDGG)

Table 725. STCU_WDGG field descriptions

Field	Description
GMBIST	Granularity of the MBIST The value of this field has to be evaluated in order to define the granularity of the MBIST run taking into account that the Watchdog timer operates at the STCU clock frequency prescaled depending on the parameter CLK_CFG defined into the STCU_CFG register and the minimum value is the length of a fixed prescaler (10 bit). The following relation gives the system clock cycles granularity of MBIST: $GMBIST_{cycles} = 2^{GMBIST} \times 1024$ The following example shows the granularity: 000b => 1 K STCU clock cycles 001b => 2 K STCU clock cycles 111b => 128 K STCU clock cycles

39.5.3.4 STCU CRC Expected Status Register (STCU_CRCE)

The STCU_CRCE register holds the expected signature of the CRC-32 loaded by the SSCM. The Self Checker computes the CRC signature of the STCU's critical signals. If the CRC computation does not match the expected value, the STCU_ERR[CRCS] bit is set.

Offset: 0x0014 Access: User read/write

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

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

Figure 826. STCU CRC Expected Status Register (STCU_CRCE)

Table 726. STCU_CRCE field descriptions

Field	Description
CRCE	CRC expected signature

39.5.3.5 STCU CRC Read Status Register (STCU_CRCR)

The STCU_CRCR register reports the value obtained by the Self Checker at the end of the self test. It can be used for diagnostics and as an additional check with respect to the STCU_ERR[CRCS] bit.

Offset: 0x0018 Access: User read/write

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

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

Figure 827. STCU CRC Read Status Register (STCU_CRCR)

Table 727. STCU_CRCR field descriptions

Field	Description
CRCR	Read CRC signature

39.5.3.6 STCU Error Register (STCU_ERR)

The STCU_ERR register includes the status flags for the STCU internal error conditions that occurred during the configuration or the self test and defines their associated fault mapping (SIR and CF).

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	WDTOSFM	CRCSSFM	ENGESFM	INVPSFM	0	0	0	0	WDTOCFM	CRCSCFM	ENGECFM	INVPCFM
W																
Reset	0	0	0	0	0	0	0	0	0	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	CFSF	NCFSF	SIRSF	0	0	0	0	WDTO	CRCS	ENGE	INVP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 828. STCU Error Register (STCU_ERR)

Table 728. STCU_ERR field descriptions

Field	Description
WDTOSFM	Watchdog timeout SIR Fault Mapping 0: Non SIR Fault mapping 1: SIR Fault mapping
CRCSSFM	CRC SIR Fault Mapping 0: Non SIR Fault mapping 1: SIR Fault mapping
ENGESFM	Engine Error SIR Fault Mapping 0: Non SIR Fault mapping 1: SIR Fault mapping
INVPSFM	Invalid Pointer SIR Fault Mapping 0: Non SIR Fault mapping 1: SIR Fault mapping
WDTOCFM	Watchdog timeout critical fault mapping 0: Non Critical Fault mapping 1: Critical Fault mapping
CRCSCFM	CRC Status critical fault mapping 0: Non critical fault mapping 1: Critical fault mapping
ENGECFM	Engine Error critical fault mapping 0: Non Critical fault mapping 1: Critical fault mapping
INVPCFM	Invalid Pointer critical fault mapping 0: Non critical fault mapping 1: Critical fault mapping
CFSF	Critical Faults Status Flag This flag reports the global status of the CF. 0: No errors that trigger the CF condition occurred. 1: At least one error that triggers the CF condition occurred.

Table 728. STCU_ERR field descriptions (continued)

Field	Description
NCFSF	Non Critical Faults Status Flag 0: No errors that trigger the NCF condition occurred. 1: At least one error that triggers the NCF condition occurred.
SIRSF	Stay In Reset Faults Status Flag 0: No Errors which trigger the SIR condition 1: There are Errors which trigger the SIR condition In the typical condition, it should not be possible to read the content of this register when this bit is set because the system should be permanently under reset. However, for diagnostic purposes, the system could exit from reset to allow SW to check the flag and attempt to trace the failure.
WDTO	Watchdog timeout 0: The self test completed within the assigned watchdog time. 1: The self test did not complete within the assigned watchdog time. This bit is also set when the STCU is activated but the self test is not run.
CRCS	CRC status 0: Successful CRC comparison 1: Failed CRC comparison
ENGE	Engine Error 0: Valid Engine execution 1: Invalid Engine execution
INVP	Invalid pointer 0: Valid linked pointer list 1: Invalid linked pointer list. The following conditions set this bit: – Initial MBIST pointer is out of range

39.5.3.7 STCU Error Key Register (STCU_ERRK)

The STCU_ERRK register implements the security key code to access to the STCU_ERR register. To write the STCU_ERR register, the SW has to:

- Write the keyx into the STCU_ERRK
- Set/Clear the STCU_ERR register

where:

- Key1 allows to clear the bit at 1
- Key2 allows to set the bit at 0

In case of invalid access, a transfer error on the IPS or SSCM bus is asserted (It depends on the selected bus) and the Key is cleared. To unlock the set/clear operation on the STCU_ERR register, the Key1 or Key2 has to be applied again.

Only one access mode (set/clear) at the time is allowed. The last key written into this register defines the access mode.

In case the STCU register access last more cycles than the one defined into the Hard-coded WDG time-out or there is a transfer error or the IPS or SSCM bus operation performed just after the Key1/Key2 Keys has been written into STCU_ERRK is not a write operation into the STCU_ERR register, the key is cleared.

The STCU_ERRK register is not readable. The value 00000000h is always returned in case of read operation.

Offset: 0x0020 Access: User write-only

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

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

Figure 829. STCU Error Key Register (STCU_ERRK)

Table 729. STCU_ERRK field descriptions

Field	Description
ERR_SK	STCU_ERRK security key = F1759034h: Key1 to reset the STCU_ERR bits at 1 = 9531B0C6h: Key2 to set the STCU_ERR bits at 0

39.5.3.8 STCU MBIST Status Low Register (STCU_MBSL)

The STCU_MBSL register includes the results corresponding to the execution of each MBIST. The STCU_MBSL register is automatically set following the completion of the MBIST run.

Offset: 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBS31	MBS30	MBS29	MBS28	MBS27	MBS26	MBS25	MBS24	MBS23	MBS22	MBS21	MBS20	MBS19	MBS18	MBS17	MBS16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBS15	MBS14	MBS13	MBS12	MBS11	MBS10	MBS9	MBS8	MBS7	MBS6	MBS5	MBS4	MBS3	MBS2	MBS1	MBS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 830. STCU MBIST Status Low Register (STCU_MBSL)

Table 730. STCU_MBSL field descriptions

Field	Description
MBSx	MBIST status 0: Failed MBIST execution 1: No Fault detected during the “x (from MBx) +1 MBIST” execution or until watchdog time out

39.5.3.9 STCU MBIST Status High Register (STCU_MBSH)

The STCU_MBSH register includes the results corresponding to the execution of each MBIST. The STCU_MBSH register is automatically set following the completion of the MBIST run.

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	MBS39	MBS38	MBS37	MBS36	MBS35	MBS34	MBS33	MBS32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 831. STCU MBIST Status High Register (STCU_MBSH)

Table 731. STCU_MBSH field descriptions

Field	Description
MBSx	MBIST status 0: Failed MBIST execution 1: No Fault detected during the “x (from MBx) +1 MBIST” execution or until watchdog time out

Table 732 shows STCU MBIST status bits to partition mapping.

Table 732. STCU MBIST status bits to partition mapping

STCU registers: STCU_MBSL./H STCU_MBEL/H STCU_MBCFML/H STCU_MBSFML/H	Corresponding memory cluster in the device	Note
MBx0	FlexCAN 0 Message buffers	—
MBx1	FlexCAN 1 Message buffers	—
MBx2	FlexCAN 2 Message buffers	—
MBx3	FlexCAN 3 Message buffers	—
MBx4	FlexCAN 4 Message buffers	—
MBx5	FlexCAN 5 Message buffers	—
MBx6	FlexCAN 0 Rx masks	—
MBx7	FlexCAN 1 Rx masks	—
MBx8	FlexCAN 2 Rx masks	—
MBx9	FlexCAN 3 Rx masks	—
MBx10	FlexCAN 4 Rx masks	—
MBx11	FlexCAN 5 Rx masks	—
MBx12	FlexRAY Protocol Engine (PE) data memory	—
MBx13	FlexRAY Contoller Host Interface (CHI) look up table	—
MBx14	FEC Tx/Rx FIFO	—
MBx15	FEC Message Information Block (MIB)	—
MBx16	e200z4d Instruction cache TAG	—
MBx17	e200z4d Instruction cache TAG	—
MBx18	e200z4d Instruction cache TAG	—
MBx19	e200z4d Instruction cache TAG	—
MBx20	e200z4d Instruction cache array	—
MBx21	e200z4d Instruction cache array	—
MBx22	e200z4d Instruction cache array	—
MBx23	e200z4d Instruction cache array	—
MBx24	CSE RAM	CSE RAM only checked when CSE is disabled
MBx25	Flash internal RAM 0	—
MBx26	Flash internal RAM 1	—
MBx27	Flash internal RAM 2	—
MBx28	DMA RAM	—

Table 732. STCU MBIST status bits to partition mapping

STCU registers: STCU_MBSL./H STCU_MBEL/H STCU_MBCFML/H STCU_MBSFML/H	Corresponding memory cluster in the device	Note
MBx29	System RAM 8K	0x4000_0000 to 0x4000_1FFF
MBx30	System RAM 24K	0x4000_2000 to 0x4000_7FFF
MBx31	System RAM 32K	0x4000_8000 to 0x4000_FFFF
MBx32	System RAM 32K	0x4001_0000 to 0x4001_7FFF
MBx33	System RAM 32K	0x4001_8000 to 0x4001_FFFF
MBx34	System RAM 32K	0x4002_0000 to 0x4002_7FFF
MBx35	System RAM 32K	0x4002_8000 to 0x4002_FFFF
MBx36	System RAM 32K	0x4003_0000 to 0x4003_7FFF
MBx37	System RAM 32K	0x4003_8000 to 0x4003_FFFF
MBx38	Boot Access Module (ROM)	CRC signature
MBx39	FlexRAY Protocol Engine (PE) Introduction memory (ROM)	CRC signature

39.5.3.10 STCU MBIST End Flag Low Register (STCU_MBEL)

The STCU_MBEL register includes the End Flag related to the execution of each MBIST. (See *Table 732*). The STCU_MBEL register is automatically updated following the completion of the MBIST run.

Offset: 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MBE31	MBE30	MBE29	MBE28	MBE27	MBE26	MBE25	MBE24	MBE23	MBE22	MBE21	MBE20	MBE19	MBE18	MBE17	MBE16
Reset	0	0	0	0	0	0	0	0	0	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	MBE15	MBE14	MBE13	MBE12	MBE11	MBE10	MBE9	MBE8	MBE7	MBE6	MBE5	MBE4	MBE3	MBE2	MBE1	MBE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 832. STCU MBIST End Flag Low Register (STCU_MBEL)

Table 733. STCU_MBEL field descriptions

Field	Description
MBEx	MBIST End status 0: MBIST execution is not finished. 1: MBIST execution is finished.

39.5.3.11 STCU MBIST End Flag High Register (STCU_MBEH)

The STCU_MBEH register includes the End Flag related to the execution of each MBIST. (See *Table 732*). The STCU_MBEH register is automatically updated following the completion of the MBIST run.

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	MBE39	MBE38	MBE37	MBE36	MBE35	MBE34	MBE33	MBE32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 833. STCU MBIST End Flag High Register (STCU_MBEH)

Table 734. STCU_MBEH field descriptions

Field	Description
MBEx	MBIST End status 0: MBIST execution is not finished. 1: MBIST execution is finished.

39.5.3.12 STCU MBIST Status-End Key Register (STCU_MBSEK)

The STCU_MBSEK register implements the key to access the STCU_MBSH/L and STCU_MBEH/L registers. To set or clear the STCU_MBSH/L and/or the STCU_MBEH/L registers the SW has to:

- Write the key into the STCU_MBSEK register
- Set/Clear the STCU_MBSH/L or STCU_MBEH/L registers

In case of invalid access, a transfer error on the IPS or SSCM bus is asserted (It depends on the selected bus interface) and the Key is cleared.

To unlock the set/clear operation on the STCU_MBSH/L or STCU_MBEH/L registers the Key has to be applied again.

In case the STCU access last more cycles than the one defined into the Hard-coded WDG timeout or there is a transfer error or the IPS or SSCM bus operation performed just after the Key has been written into STCU_MBSEK is not a write operation into STCU_MBSH/L or STCU_MBEH/L registers, the key is cleared.

The STCU_MBSEK register is not readable, a 0x00000000 value is always returned in case of read operation.

The STCU_MBEH register includes the End Flag related to the execution of each MBIST. The STCU_MBEH register is automatically updated following the completion of the MBIST run.

Offset: 0x004C Access: User write-only

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

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

Figure 834. STCU MBIST Status-End Key Register (STCU_MBSEK)

Table 735. STCU_MBSEK field descriptions

Field	Description
MBSEK	STCU_MBSH/L and STCU_MBEH/L register key = 825A132Bh: Key for a write operation

39.5.3.13 STCU MBIST Critical FM Low Register (STCU_MBCFML)

The STCU_MBCFML register defines the MBIST fault mapping in terms of critical or non critical faults. (See *Table 732*). The STCU_MBCFML register is automatically set following the completion of the MBIST run.

Offset: 0x0050 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MBCFM31	MBCFM30	MBCFM29	MBCFM28	MBCFM27	MBCFM26	MBCFM25	MBCFM24	MBCFM23	MBCFM22	MBCFM21	MBCFM20	MBCFM19	MBCFM18	MBCFM17	MBCFM16
Reset	0	0	0	0	0	0	0	0	0	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	MBCFM15	MBCFM14	MBCFM13	MBCFM12	MBCFM11	MBCFM10	MBCFM9	MBCFM8	MBCFM7	MBCFM6	MBCFM5	MBCFM4	MBCFM3	MBCFM2	MBCFM1	MBCFM0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 835. STCU MBIST Critical FM Low Register (STCU_MBCFML)

Table 736. STCU_MBCFML field descriptions

Field	Description
MBCFMx	MBIST critical fault mapping 0: This MBIST is a NCF. 1: This MBIST is a CF.

39.5.3.14 STCU MBIST Critical FM High Register (STCU_MBCFMH)

The STCU_MBCFMH register defines the MBIST fault mapping in terms of critical or non critical faults. (See *Table 732*). The STCU_MBCFMH register is automatically set following the completion of the MBIST run.

Offset: 0x0054 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	MBCFM39	MBCFM38	MBCFM37	MBCFM36	MBCFM35	MBCFM34	MBCFM33	MBCFM32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 836. STCU MBIST Critical FM High Register (STCU_MBCFMH)

Table 737. STCU_MBCFMH field descriptions

Field	Description
MBCFMx	MBIST critical fault mapping 0: This MBIST is a NCF. 1: This MBIST is a CF.

39.5.3.15 STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML)

The STCU_MBSFML register defines the MBIST fault mapping in terms of the SIR condition. The STCU_MBSFML register is automatically set following the completion of the MBIST run.

Offset: 0x0058 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MBSFM31	MBSFM30	MBSFM29	MBSFM28	MBSFM27	MBSFM26	MBSFM25	MBSFM24	MBSFM23	MBSFM22	MBSFM21	MBSFM20	MBSFM19	MBSFM18	MBSFM17	MBSFM16
Reset	0	0	0	0	0	0	0	0	0	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	MBSFM15	MBSFM14	MBSFM13	MBSFM12	MBSFM11	MBSFM10	MBSFM9	MBSFM8	MBSFM7	MBSFM6	MBSFM5	MBSFM4	MBSFM3	MBSFM2	MBSFM1	MBSFM0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 837. STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML)

Table 738. STCU_MBSFML field descriptions

Field	Description
MBSFMx	MBIST SIR Fault Mapping 0: This MBIST is not a SIR fault. 1: This MBIST is a SIR fault.

39.5.3.16 STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)

The STCU_MBSFMH register defines the MBIST fault mapping in terms of the SIR condition. The STCU_MBSFMH register is automatically set following the completion of the MBIST run.

Offset: 0x005C 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	MBSFM39	MBSFM38	MBSFM37	MBSFM36	MBSFM35	MBSFM34	MBSFM33	MBSFM32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 838. STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)

Table 739. STCU_MBSFMH field descriptions

Field	Description
MBSFMx	MBIST SIR Fault Mapping 0: This MBIST is not a SIR fault. 1: This MBIST is a SIR fault.

39.5.3.17 STCU MBIST FM Key Register (STCU_MBFMK)

The STCU_MBFMK register implements the key to access the STCU_MBCFML/H and STCU_MBSFML/H registers.

To set/clear the STCU_MBCFML/H and/or STCU_MBSFML/H registers the SW has to:

- Write the key into the STCU_MBFMK register
- Set/Clear the STCU_MBCFML/H or STCU_MBSFML/H registers

In case of invalid access, a transfer error on the IPS or SSCM bus is asserted (It depends on the selected bus interface) and the Key is cleared.

To unlock the set/clear operation on the STCU_MBCFML/H or STCU_MBSFML/H registers the Key has to be applied again.

In case the STCU register access last more cycles than the one defined into the Hard-coded WDG time-out or there is a transfer error or the IPS or SSCM bus operation performed just after the Key has been written into STCU_MBFMK is not a write operation into the STCU_MBCFML/H or STCU_MBSFML/H registers, the key is cleared.

The STCU_MBFMK register is not readable, a 0x00000000 value is always returned in case of read operation.

Offset: 0x0060 Access: User write-only

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

Figure 839. STCU MBIST FM Key Register (STCU_MBFMK)

Table 740. STCU_MBFMK field descriptions

Field	Description
MBFMK	STCU_MBCFML/H and STCU_MBSFML/H registers key = 751AC490h: Key for a write operation

39.5.3.18 STCU MBIST Control Register (STCU_MB_CTRL)

The STCU_MB_CTRL registers define the control fields of each MBIST.

Offset: 0x0300 + ((k-1) × 0x4)[1] Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSM	PTR							0	0	MB_TIME					
W																
Reset	0/1[2]	0/1[2]	0/1[2]	0/1[2]	0/1[2]	0/1[2]	0/1[2]	0/1[2]	0	0	0/1[3]	0/1[3]	0/1[3]	0/1[3]	0/1[3]	0/1[3]

	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

Figure 840. STCU MBIST Control Register (STCU_MB_CTRL)

1. k = 1 to 40.
2. Reset value of CSM and PTR for k= 1 to 39: CSM=1, PTR=k+0x10.Rest value for k=40 : CSM= 0 and PTR=0x7f
3. For Reset values , please refer *Table 721*

Table 741. STCU_MB_CTRL field descriptions

Field	Description
CSM	Concurrent/sequential mode. 0 Sequential mode. 1 Concurrent mode.
PTR	next MBIST pointer PTR defines the logical pointer to the next MBIST to be scheduled. When the NIL pointer is encountered, the self testing procedure is stopped when the current MBIST has been completed. 10h to (10h + (k-1): pointer to MBIST and k ranges from 1 to 40 7Fh: pointer to NIL. No BIST execution. others: invalid pointer => an error is set into the STCU_ERR register.
MB_TIME	Memory BIST RUN Time The time budget of the MBIST is evaluated applying the following relation: $MB_{cycles} = MBTIME \times GMBIST_{cycles}$ In case the MBIST is not completed within the MB_TIME the STCU_ERR[WDTO] bit is set.

39.5.4 Self-Test sequence after reset trigger

This is the typical mode of using the STCU module after reset trigger event is applied to STCU. The SSCM DCF bus is used to retrieve the STCU schedule and MBIST execution parameters stored into the NVM memory.

The target is to cover the amount of physical defect into the System RAMs/ROMs of the System. The suggested sequence is the following:

- Program the STCU_CFG register in order to: program the core and MBIST TCK clock prescaling factor setting the CLK_CFG bits and set the pointer to the first MBIST to be executed.
- Unlock the STCU_MBCFML/H and STCU_MBSFML/H access writing the correct key into the STCU_MBFMK register and set into these registers the specified CF/NCF/SIR condition.
- Unlock the STCU_MBSL/H and STCU_MBEL/H registers access writing the correct key into the STCU_MBSEK register and set into these registers respectively the Status and End Bits of the NOT RUN MBIST.
- Program the STCU_MB_CTRL registers of each MBIST from 1 to 40 to be executed.
- Program the GMBIST granularity fields into the STCU_WDGG register.
- Program the expected CRCE value expected at the end of the off-line Self Test sequence into the STCU_CRCE register.
- MBIST execution starts.
- After execution is completed:
  - In case SIR condition is detected, the STCU keep the system in reset.
  - In case CF condition is detected, System will go to safe mode
  - In case NCF condition is detected, an interrupt will be generated.

40 Cryptographic Services Engine (CSE)

40.1 Introduction

40.1.1 Overview

The Cryptographic Services Engine (CSE) is a peripheral module that implements the security functions described in the *Secure Hardware Extension (SHE) Functional Specification Version 1.1.* The CSE design includes a host interface with a set of memory mapped registers that are used by the CPU to issue commands and a system bus interface that allows the CSE to directly access system memory. Two dedicated blocks of system Flash memory are used by the CSE for secure key storage.

40.1.2 Features

The CSE has the following features:

- Secure storage for cryptographic keys.
- AES-128 encryption and decryption.
- AES-128 CMAC authentication.
- Random number generation.
- Secure boot mode.
- System bus master interface.

40.1.3 Modes of operation

The CSE supports operation in normal and debug modes of operation. The use of the cryptographic keys stored by the CSE is controlled based on the activation of the CPU debug port and the successful completion of the secure boot process.

The CSE has a low power mode which disables the clock to all logic except the host interface. Register accesses are supported in this mode but commands are not processed.

40.1.4 Block diagram

The CSE design includes a command processor, host interface, system bus interface, local memory, AES logic and True Random Number Generator (TRNG) as shown in *Figure 841*.

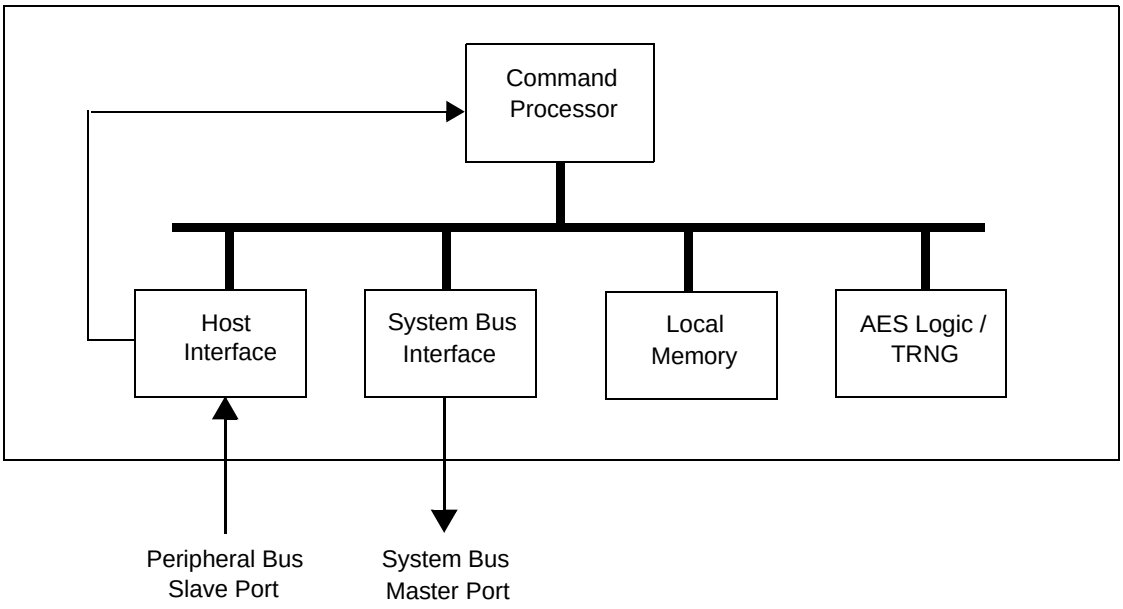


Figure 841. CSE block diagram

40.2 External signal description

The CSE has no external interface signals.

40.3 Memory map and register definition

The CSE programming model has ten 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid.

40.3.1 Memory map

The CSE memory map is shown in *Table 742*.

Table 742. CSE memory map

Base address: 0xFFF1_C000		
Address offset	**Register**	**Location**
0x0000	CSE Control Register (CSE_CR)	*on page 1334*
0x0004	CSE Status Register (CSE_SR)	*on page 1335*
0x0008	CSE Interrupt Register (CSE_IR)	*on page 1336*
0x000C	CSE Error Code Register (CSE_ECR)	*on page 1337*

Table 742. CSE memory map (continued)

Base address: 0xFFF1_C000		
Address offset	**Register**	**Location**
0x0010–0x001C	Reserved	
0x0020	CSE Command Register (CSE_CMD)	*on page 1338*
0x0024	CSE Parameter 1 Register (CSE_P1)	*on page 1339*
0x0028	CSE Parameter 2 Register (CSE_P2)	*on page 1339*
0x002C	CSE Parameter 3 Register (CSE_P3)	*on page 1339*
0x0030	CSE Parameter 4 Register (CSE_P4)	*on page 1339*
0x0034	CSE Parameter 5 Register (CSE_P5)	*on page 1339*
0x0038–0x3FFF	Reserved	

40.3.2 Register descriptions

The following sections detail the individual registers within the CSE programming model.

40.3.2.1 CSE Control Register (CSE_CR)

The CSE_CR contains fields for configuring and controlling operation of the CSE.

Offset 0x0000 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	DIV								0	0	0	0	MDIS	SUS	0	CIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0(1)

Figure 842. CSE Control Register (CSE_CR)

1. The reset value of the CIE bit is 0 on POR, but when the platform reset is lifted and CSE execution starts, the value of this bit is updated to 1. So, after the first read access to this bit, the value read back is 1.

Table 743. CSE_CR field descriptions

Field	Description
DIV	TRNG Clock Divider Select. The DIV field sets the clock divide ratio for the TRNG clock. The TRNG clock is the system clock divided by a programmable ratio: TRNG Clk Freq = Sys Clk Freq / (2*(DIV+1)) The divide ratio must be set such that the TRNG clock frequency is between 500 kHz and 2 MHz. 0x00 = divide by 2 0x01 = divide by 4 0x02 = divide by 6 .... 0xFE = divide by 510 0xFF = divide by 512
MDIS	Module Disable. When the MDIS bit is set, the CSE is put into a low power mode which disables the clock to all of the CSE logic except for the host interface. The MDIS bit should not be set during command processing (CSE_SR[BSY]=1). The current command should be canceled and processing stopped (CSE_SR[BSY]=0) before setting the MDIS bit. 0 = Normal mode. 1 = Low power mode.
SUS	Suspend command processing. When the SUS bit is set, the CSE suspends processing of the current command until the SUS bit is cleared. The current execution status of the command processor is reflected by the CSE_SR[EX] flag. 0 = Enable processing of commands. 1 = Suspend command processing.
CIE	Command Complete Interrupt Enable. When the CIE bit is set, an interrupt request is generated and the CSE_IR[CIF] flag is set when command processing is completed. 0 = Command complete interrupt disabled 1 = Command complete Interrupt enabled

40.3.2.2 CSE Status Register (CSE_SR)

The CSE_SR contains flags indicating the status of the CSE. Reading CSE_SR[24:31] is the same as the SHE GET_STATUS command.

Offset 0x0004 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	0	0	0	0	0	0	0	EX	IDB	EDB	RIN	BOK	BFN	BIN	SB	BSY
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 843. CSE Status Register (CSE_SR)

Table 744. CSE_SR field descriptions

Field	Description
EX	Execute. 0 = Command processor is idle or suspended. 1 = Command processor is executing a command.
IDB	Internal Debug. The IDB bit is set by the SECURE_BOOT, INIT_CSE and DEBUG_AUTH commands when no user keys are stored. It is cleared on reset and by the LOAD_KEY command. 0 = Internal debug functions disabled. 1 = Internal debug functions enabled.
EDB	External Debug. The EDB bit is set when the CPU debug port is activated and is cleared on reset. 0 = External debugger not attached. 1 = External debugger attached.
RIN	Random Number Generator Initialized. The RIN bit is set by the INIT_RNG command and is cleared on reset and by the DEBUG_AUTH command. 0 = Random number generator not initialized. 1 = Random number generator initialized.
BOK	Secure Boot OK. The BOK bit is set by the successful completion of the SECURE_BOOT command. It is cleared by reset and by the BOOT_FAILURE command. 0 = Secure boot not completed or secure boot failure. 1 = Secure boot successful.
BFN	Secure Boot Finished. The BFN bit is set by the SECURE_BOOT command when the BIN bit is set, an error is encountered or the BOOT_MAC value does not match. It is also set by the BOOT_OK and not set in case that SECURE_BOOT command finishes with a flash error (CSE_ECR 0x13) due to illegal firmware. It is cleared on reset. 0 = Secure boot not finished. 1 = Secure boot finished.
BIN	Secure Boot Initialization. The BIN bit is set by the SECURE_BOOT command if the BOOT_MAC memory slot is empty and is cleared on reset. 0 = Secure boot personalization not completed. 1 = Secure boot personalization completed.
SB	Secure Boot. The SB bit is set by the SECURE _BOOT command if the BOOT_MAC_KEY slot is not empty, and is cleared on reset. 0 = Secure boot not activated. 1 = Secure boot activated.
BSY	Busy. The BSY bit is set when a command is issued and cleared when command processing is completed. 0 = Command processing completed. 1 = Command processing not completed.

40.3.2.3 CSE Interrupt Register (CSE_IR)

The CSE_IR contains the command completion interrupt flag bit.

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	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(1)

Figure 844. CSE Interrupt Register (CSE_IR)

1. The reset value on POR for this bit is 0. But, when the device boots up in a mode that supports SECURE_BOOT, the value for this bit is updated to 1 on the command completion.

Table 745. CSE_IR field descriptions

Field	Description
CIF	Command Complete Interrupt Flag. The CIF flag reflects the state of the command complete interrupt request. The CIF flag and interrupt request are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 = No interrupt request. 1 = Interrupt request due to completion of a command.

40.3.2.4 CSE Error Code Register (CSE_ECR)

The CSE_ECR contains the error code from the last completed command.

Offset 0x000C 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	0	0	0	0	0	0	0	0	0	0	0	EC				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 845. CSE Error Code Register (CSE_ECR)

Table 746. CSE_ECR field descriptions

Field	Description
EC	Error Code from the last command completed. 0x00 = No error. 0x02 = Command sequence error. 0x03 = Key not available. 0x04 = Invalid key. 0x05 = Empty key. 0x06 = No secure boot. 0x07 = Key write protected. 0x08 = Key update error. 0x09 = Random number seed not initialized. 0x0A = Internal debug not allowed. 0x0B = Command issued while busy. 0x0C = System memory error. 0x10 = Internal memory error. 0x11 = Invalid command. 0x12 = TRNG error. 0x13 = CSE Flash block error. 0x14 = Internal command processor error. 0x15 = Length error.

40.3.2.5 CSE Command Register (CSE_CMD)

Commands are issued by first loading the appropriate parameter registers (CSE_Px) and then writing a command code to the command register (CSE_CMD). Reads of the CSE_CMD register have no effect on the operation of the CSE. See *Section 40.4.2: Command Processing* for a description of command processing and *Section 40.5: CSE Commands* for a description of each command.

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	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	CMD				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 846. CSE Command Register (CSE_CMD)

Table 747. CSE_CMD field descriptions

Field	Description
CMD	Command. See *Section 40.5: CSE Commands* for details of each command. 0x01 = ENC_ECB 0x02 = ENC_CBC 0x03 = DEC_ECB 0x04 = DEC_CBC 0x05 = GENERATE_MAC 0x06 = VERIFY_MAC 0x07 = LOAD_KEY 0x08 = LOAD_PLAIN_KEY 0x09 = EXPORT_RAM_KEY 0x0A = INIT_RNG 0x0B = EXTEND_SEED 0x0C = RND 0x0D = SECURE_BOOT 0x0E = BOOT_FAILURE 0x0F = BOOT_OK 0x10 = GET_ID 0x11 = CANCEL 0x12 = DEBUG_CHAL 0x13 = DEBUG_AUTH 0x14 = TRNG_RND 0x15 = INIT_CSE

40.3.2.6 CSE Command Parameter Registers (CSE_Px)

The five parameter registers (CSE_Px) are used to hold command parameter values. See *Section 40.4.2: Command Processing* for a description of how to issue commands and *Section 40.5: CSE Commands* for a description of each command. The parameter registers are read only when the CSE_SR[BSY] bit is set.

Offset 0x0020 + 4*x 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 W	PARM																															
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 847. CSE Parameter Register (CSE_Px)

1. The reset value out of POR is 0, but when the device boots up from Flash, the registers reflect the values based on the command execution. For details, see *Table 764*.

Table 748. CSE_Px field descriptions

Field	Description
PARM	Command parameter (data value or address of data value).

40.4 CSE functional description

The CSE implements a comprehensive set of cryptographic functions as described in the *SHE Functional Specification* including secure key storage, AES encryption, secure boot, AES CMAC authentication and random number generation.

40.4.1 Host Interface

The host interface includes all of the CSE memory mapped registers (see *Section 40.3: Memory map and register definition*).

The Control Register (CSE_CR) is used to set configuration options and control operation of the CSE. If the CSE_CR[CIE] bit is set, an interrupt request is generated after the CSE finishes processing each command. The interrupt is cleared by writing a one to the CSE_IR[CIF] bit. When the CSE_CR[SUS] bit is set, the CSE suspends processing of the current command and does not respond to new commands including the CANCEL command. Command processing is resumed by clearing the CSE_CR[SUS] bit. The CSE can not respond immediately to changes in the CSE_CR[SUS] bit so the CSE_SR[EX] flag indicates the current status of the CSE command processor. The CSE_SR[EX] flag is set when the CSE command processor is running and is cleared when the CSE is idle or has suspended processing. The CSE_CR[MDIS] bit puts the CSE into a low power mode. In the low power mode, registers can be accessed but commands are not processed. Before setting the CSE_CR[MDIS] bit or entering any device low power modes, application software should cancel any pending command and wait for the CSE_SR[BSY] bit to be cleared.

The Status Register (CSE_SR) contains a set of nine status flags. Reading CSE_SR[24:31] is equivalent to the SHE GET_STATUS command. The CSE_SR[BSY] bit indicates when the CSE is processing a command. The flag is cleared when processing is completed. Polling the CSE_SR[BSY] bit is an alternative to interrupt driven operation. The CSE_SR[SB], CSE_SR[BIN], CSE_SR[BFN] and CSE_SR[BOK] bits reflect the secure boot status (see *Section 40.4.6: Secure Boot*). The CSE_SR[RIN] bit is set when the PRNG is initialized with a seed value generated by the TRNG. The CSE_SR[EDB] is set if an external debugger is connected. The CSE_SR[IDB] flag is set when internal debugging has been enabled with the DEBUG_CHAL and DEBUG_AUTH commands. The CSE_SR is a read only register and can be read at any time.

The Error Code Register (CSE_ECR) contains the error code from the last completed command. The CSE_ECR is set to zero if no error occurred while processing the command. The CSE_ECR is a read only register.

40.4.2 Command Processing

CSE functions are accessed through a set of commands which are listed in *Section 40.5: CSE Commands*. Commands are issued by first loading the appropriate parameter registers (CSE_Px) and then writing a command code to the command register (CSE_CMD). CSE_CMD register reads have no effect on the operation of the CSE.

Command inputs and outputs that are 32 bits or less in size are stored directly in a parameter register. Inputs and outputs that are larger than 32 bits are kept in system memory and the address of the data (pointer) is loaded into the parameter register. All data must be aligned on a 32-bit (word) boundary. The two least significant bits of an address are ignored. All data is stored in big-endian format with the most significant byte of the data block stored in the lowest address. On devices with a 64-bit system bus, the CSE will perform double word (64-bit) reads and writes to system memory if the data is aligned on a double word boundary.

The first command issued to the CSE after reset must be either SECURE_BOOT or INIT_CSE. In some device boot modes, system boot logic or BAM code will issue the SECURE_BOOT command which initializes the CSE and starts the SHE secure boot protocol (see *Section 40.4.6: Secure Boot*). For the other boot modes, application software must issue the INIT_CSE command before issuing other CSE commands. The INIT_CSE command can only be issued one time after reset and only in boot modes that do not issue the SECURE_BOOT command. Application software cannot issue the SECURE_BOOT command.

The CSE can only process one command at a time. Commands take a number of clock cycles to complete so either the CSE_SR[BSY] flag, the interrupt request needs to be used to determine when the output data is valid and the next command can be issued. Parameter registers cannot be modified while the CSE is processing a command. Command processing can be suspended and resumed using the CSE_CR[SUS] bit (see *Section 40.4.1: Host Interface*). If the CSE_CMD register is written while a command is being processed, processing is aborted and an error is generated with CSE_ECR[EC] = 0x0B. The CANCEL command can be used to abort processing of a command without generating an error. The CSE_ECR is updated and the CSE_SR[BSY] bit is cleared upon completion of command processing (see *Section 40.4.8: Error Handling*).

40.4.3 Secure Storage

The CSE provides secure, non-volatile storage for cryptographic keys as described in the *SHE Functional Specification*. The keys are stored in 15 memory slots, with one ROM slot, 13 non-volatile slots and one RAM slot as shown in *Table 749*. The first four slots have a dedicated use, the other slots are available for application specific keys. The BOOT_MAC slot is loaded with a MAC value used by the secure boot process. All other slots are used for encryption or message authentication keys. The SECRET_KEY slot is programmed with a random value during device fabrication. All CSE encryption and message authentication commands specify a key by its Key ID.

Table 749. Memory Slots

Slot Name	Key ID	Type
SECRET_KEY	0x0	ROM
MASTER_ECU_KEY	0x1	non-volatile
BOOT_MAC_KEY	0x2	non-volatile
BOOT_MAC	0x3	non-volatile
KEY_1	0x4	non-volatile
KEY_2	0x5	non-volatile
KEY_3	0x6	non-volatile
KEY_4	0x7	non-volatile
KEY_5	0x8	non-volatile
KEY_6	0x9	non-volatile
KEY_7	0xA	non-volatile
KEY_8	0xB	non-volatile

Table 749. Memory Slots

Slot Name	Key ID	Type
KEY_9	0xC	non-volatile
KEY_10	0xD	non-volatile
RAM_KEY	0xE	RAM

In addition to the 15 memory slots, the CSE holds a 120-bit read only unique identification number (UID) which is programmed during device fabrication. The UID is used in the memory update procedure and is available for application specific uses.

Each memory slot holds a 128-bit value, five security flags and a 28-bit counter. The security flags are defined in *Table 750*. The 28-bit counter must be advanced each time a new key is loaded. The counter value is zero for an empty memory slot.

Table 750. Memory Slot Security Flags

Flag Name	Description
WRITE_PROT	If set, the memory slot can not be updated.
BOOT_PROT	If set, the memory slot is disabled if CSE_SR[BOK] = 0 or CSE_SR[BFN] = 0.
DEBUG_PROT	If set, the memory slot is disabled if CSE_SR[EDB] = 1.
KEY_USAGE	If set, the memory slot holds a MAC key, otherwise it holds an encryption key.
WILDCARD	If set, the memory slot can not be updated with the wildcard UID.

The memory slot security flags for a specified key are checked against the associated bits in the CSE_SR during command processing as described in the SHE specification. If the memory slot is disabled due to security flag settings, a key not available error (EC=0x03) is returned by the command.

A key is loaded into a memory slot using the LOAD_KEY command which implements the SHE memory update protocol. An empty key slot can only be used as the authorization key to update itself (KeyID=AuthID). In this case, the authorization key has a value of zero.

The BOOT_PROT and DEBUG_PROT security flags are not enforced for the LOAD_KEY command except when the RAM_KEY slot is loaded. In this case, the BOOT_PROT and DEBUG_PROT flags of the AuthID slot are applied. Additionally, when loading the RAM_KEY slot, the counter and security flag fields in the M2 message must be zero.

The LOAD_PLAIN_KEY command can be used to load a key into the RAM_KEY slot. The EXPORT_RAM_KEY command can then be used to export the key in an encrypted form compatible with the LOAD_KEY command for storage outside the CSE. The BOOT_PROT and DEBUG_PROT security flags of the MASTER_ECU_KEY are enforced for the EXPORT_RAM_KEY command.

Two dedicated blocks in the system Flash memory are used to implement the secure storage feature. These blocks are not program visible and only the CSE can read, erase and program these blocks. The CSE reads and writes to the system Flash memory via the system bus master interface during the SECURE_BOOT, INIT_CSE, DEBUG_AUTH and LOAD_KEY commands. During execution of these commands, the system MPU (if present) must be configured to allow the CSE access to the system Flash memory and other bus masters must be programmed properly to avoid interfering with the CSE.

40.4.4 Encryption and Decryption

The CSE supports AES-128 encryption and decryption in ECB and CBC modes of operation. The key is selected from one of the memory slots which must be enabled for the operation. A plaintext key can be loaded into the RAM_KEY slot using the LOAD_PLAIN_KEY command for keys that are not stored in a non-volatile memory slot.

40.4.5 Message Authentication

The CSE uses the AES-128 CMAC algorithm for message authentication. The key for the CMAC operation is selected from one of the memory slots which must be enabled for the operation. A plaintext key can be loaded into the RAM_KEY slot using the LOAD_PLAIN_KEY command for keys that are not stored in a non-volatile memory slot. The VERIFY_MAC command supports comparison of a calculated MAC with an input MAC value.

40.4.6 Secure Boot

The CSE implements the SHE secure boot protocol. When CSE is supported, the SSCM logic issues the SECURE_BOOT command to the CSE which starts the secure boot process. The first step in the process is for the CSE to download the command processor firmware and memory slot data from the CSE Flash blocks into local memory. If the BOOT_MAC_KEY slot is empty, the CSE_SR[SB] flag is cleared and the process is finished. Otherwise, the CSE_SR[SB] flag is set and CSE calculates the MAC over the specified bootloader code. If the BOOT_MAC slot is empty, the calculated MAC is loaded into the BOOT_MAC slot, the CSE_SR[BIN] and CSE_SR[BFN] flags are set and the process is finished. Otherwise, the calculated MAC value is compared to the value in the BOOT_MAC slot. If the values match, the CSE_SR[BOK] bit is set. Otherwise, the CSE_SR[BFN] bit is set. If the CSE_SR[BOK] flag is set, the user boot code can issue the BOOK_OK command which sets the CSE_SR[BFN] bit. The memory slots which have the BOOT_PROT flag set are enabled when both the CSE_SR[BFN] and CSE_SR[BOK] flags are set.

The SECURE_BOOT command can run either before the user boot code is executed or in parallel with the user boot code. The secure boot mode option is selected using the NVUSRO_1[CSE_RUN_MODE] user option bit (see *Section 35.2.5.28: Nonvolatile User Options register 1(NVUSRO_1)*.) If the parallel mode is used, the user boot code must suspend command processing (CSE_CR[SUS] bit set) while performing any operations (such as configuring the Flash controller) that may interfere with the CSE accessing flash memory.

The CSE implements an internal timer that is started when the SECURE_BOOT command is issued. The initial value of the timer is set to 0x00F4_2400 (16,000,000 CSE system clocks). If the timer expires before the SECURE_BOOT command completes, any application code intended to be blocked by the SECURE_BOOT process will instead see the non-completed SECURE_BOOT status of CSE_SR[BSY]=1 and CSE_SR[BOK]=0. In the event of a SECURE_BOOT timeout, the CSE will continue executing the SECURE_BOOT command until completion, until suspended, or until receiving the CANCEL command from the application code. There is no error code associated with a SECURE_BOOT timeout.

40.4.7 Random Number Generation

The CSE has both a Pseudo Random Number Generator (PRNG) and a True Random Number Generator (TRNG). The PRNG has a 128-bit state variable and uses AES in output feedback mode to generate pseudo random values. A key derived from the SECRET_KEY is used for the PRNG. The RND command updates the state of the PRNG and returns the

128-bit random value. The EXTEND_SEED command can be used to add entropy to the PRNG state. The PRNG state is initialized after each reset with the INIT_RNG command which uses the TRNG to generate a 128-bit seed value for the PRNG. The CSE_SR[RIN] flag is set when the PRNG is initialized.

The INIT_RNG and TRNG_RND commands use the TRNG to generate truly random values. The TRNG hardware runs off of a slower clock derived from the system clock. The CSE_CR[DIV] field needs to be configured for these commands such that the TRNG clock is between 500 kHz and 2 MHz. Random values generated by the TRNG are checked with a statistical test to verify proper operation of the TRNG. If the test fails, a TRNG error (EC=0x12) is returned. Due to the statistical nature of this test, there is a very small probability ($<10^{-9}$) that a properly operating TRNG will return an error. If an TRNG error is returned, the command can be issued again.

40.4.8 Error Handling

When the CSE command processor encounters an error condition, it stops processing and returns an error code as described in *Table 751*. The CSE_SR[BSY] and CSE_SR[EX] bits are cleared with interrupt and/or DMA requests generated the same as if the command had completed successfully. In most cases, error conditions are detected before data processing begins and no output values are written. Intermediate or invalid output data is never written to the parameter registers or system memory. However, it is possible for outputs to system memory to be partially written when an error occurs. The CSE does not zero out this partially written data.

Table 751. Error Code Summary

Error Code	Error Description	Error Conditions
0x00	No Error	1. Command successfully executed with no errors encountered.
0x02	Command sequence error.	1. Command issued (except for CANCEL) before the SECURE_BOOT or INIT_CSE command. 2. The DEBUG_AUTH command issued before the DEBUG_CHAL command. 3. SECURE_BOOT or INIT_CSE command issued after an initial SECURE_BOOT or INIT_CSE is issued.
0x03	Key not available.	1. A required key is not available due to a BOOT_PROT or DEBUG_PROT security flag restriction (see *Section 40.4.3: Secure Storage*).
0x04	Invalid key.	1. The specified key slot is not valid for the command. (see the SHE specification). 2. The specified key slot is not available due to a KEY_USAGE security flag restriction (see *Section 40.4.3: Secure Storage*).
0x05	Empty key.	1. The specified key slot is empty.
0x06	No secure boot.	1. SECURE_BOOT issued when secure boot is disabled or the BOOT_MAC_KEY slot is empty. 2. BOOT_FAIL or BOOT_OK issued with incorrect settings for either the CSE_SR[SB], CSE_SR[BFN] or CSE_SR[BOK] flags.
0x07	Key write protected.	1. Attempt to load a key slot with the WRITE_PROT security flag set. 2. Attempt to enter internal debug mode (DEBUG_CHAL, DEBUG_AUTH) with a WRITE_PROT security flag set in one or more key slots.

Table 751. Error Code Summary

Error Code	Error Description	Error Conditions
0x08	Key update error.	The LOAD_KEY command failed due to one of the following conditions: 1. The M3 MAC value is invalid. 2. The wildcard UID is specified with the WILDCARD flag set. 3. The specified UID does not match the device UID. 4. The update counter or security flag values are not zero when loading the RAM_KEY slot. 5. The specified update counter value is not greater than the current counter value for the slot. (Counter value is zero for an empty slot.)
0x09	Random number seed not initialized.	1. RND, EXTEND_SEED or DEBUG_CHAL command issued before the INIT_RNG command.
0x0A	Internal debug not allowed.	1. DEBUG_AUTH command issued with invalid MAC value.
0x0B	Command issued while busy.	1. Command issued when the CSE_SR[BSY] bit is set.
0x0C	System memory error	1. A system memory error was encountered while executing the command. (The CSE Flash block error code is generated for bus errors encountered when accessing the CSE Flash blocks.)
0x10	Internal memory error.	1. An internal memory error was encountered while executing the command.
0x11	Invalid command.	1. Value written to CSE_CMD register is out of range.
0x12	TRNG error.	1. One or more statistical tests run on the TRNG output failed. (see *Section 40.5.12: Generate Random Number*)
0x13	CSE Flash block error.	1. Error reading, programming or erasing one of the CSE Flash blocks. 2. UID or SECRET_KEY required but not available.
0x14	Internal command processor error.	1. An internal error condition was encountered while executing the command.
0x15	Length error.	1. MAC length for VERIFY_MAC command is greater than 128. 2. Message length for GENERATE_MAC or VERIFY_MAC command is greater than 0x7FFFFFFF (4GB).

40.5 CSE Commands

This section describes the set of CSE commands. Commands are issued by first loading the appropriate parameter registers (CSE_Px) and then writing a command code to the command register (CSE_CMD) as described in *Section 40.4.2: Command Processing*. The first command issued to the CSE after reset must be INIT_CSE for device boot modes that do not support secure boot. Command inputs and outputs that are 32 bits or less in size are stored directly in a parameter register. Inputs and outputs that are larger than 32 bits are kept in system memory and the address of the data (pointer) is loaded into the parameter register. The data direction indication in the tables below refers to the command parameter value which may be stored in memory.

40.5.1 Encrypt ECB

The ENC_ECB command performs AES-128 encryption in ECB mode on *n* 128-bit blocks of data with the parameters specified in *Table 752*.

Table 752. ENC_ECB Command

Register	Value	Data Direction
CSE_CMD	0x01	—
CSE_P1	Key ID	Input
CSE_P2	Number of blocks (*n*)	Input
CSE_P3	First plaintext block address	Input
CSE_P4	First ciphertext block address	Output

40.5.2 Encrypt CBC

The ENC_CBC command performs AES-128 encryption on *n* 128-bit blocks of data in CBC mode with the parameters specified in *Table 753*. The number of blocks parameter is a 32-bit value.

Table 753. ENC_CBC Command

Register	Value	Data Direction
CSE_CMD	0x02	—
CSE_P1	Key ID	Input
CSE_P2	IV address	Input
CSE_P3	Number of blocks (*n*)	Input
CSE_P4	First plaintext block address	Input
CSE_P5	First ciphertext block address	Output

40.5.3 Decrypt ECB

The DEC_ECB command performs AES-128 ECB decryption on *n* 128-bit blocks of data with the parameters specified in *Table 754*.

Table 754. DEC_ECB Command

Register	Value	Data Direction
CSE_CMD	0x03	—
CSE_P1	Key ID	Input
CSE_P2	Number of blocks (*n*)	Input

Table 754. DEC_ECB Command

Register	Value	Data Direction
CSE_P3	First ciphertext block address	Input
CSE_P4	First plaintext block address	Output

40.5.4 Decrypt CBC

The DEC_CBC command performs AES-128 decryption in CBC mode on *n* 128-bit blocks of data with the parameters specified in *Table 755*. The number of blocks parameter is a 32-bit value.

Table 755. DEC_CBC Command

Register	Value	Data Direction
CSE_CMD	0x04	—
CSE_P1	Key ID	Input
CSE_P2	IV address	Input
CSE_P3	Number of blocks (*n)*	Input
CSE_P4	First ciphertext block address	Input
CSE_P5	First plaintext block address	Output

40.5.5 Generate MAC

The GENERATE_MAC command calculates the MAC of a given message with the parameters specified in *Table 756*. The AES CMAC algorithm is used to calculate a 128-bit MAC output. The message length input is a 64-bit value which specifies the length of the message in bits. A length error (EC=0x15) is returned if the message length is greater than 0x7fffffff (4GB).

Table 756. GENERATE_MAC Command

Register	Value	Data Direction
CSE_CMD	0x05	—
CSE_P1	Key ID	Input
CSE_P2	Message length (bits) address	Input
CSE_P3	Message start address	Input
CSE_P4	MAC address	Output

40.5.6 Verify MAC

The VERIFY_MAC command verifies the MAC of a given message with the parameters specified in *Table 757*. The AES CMAC algorithm is used to calculate a 128-bit MAC which is truncated according to the MAC length parameter which specifies the number of most significant bits in the MAC to compare. A MAC length value of zero indicates that all 128-bits are compared; a value greater than 128 returns a length error (EC=0x15). The message length input is a 64-bit value which specifies the length of the message in bits. A length error (EC=0x15) is returned if the message length is greater than 0x7fffffff (4GB). If the input MAC matches the MAC calculated over the message, the CSE_P5 register is set to zero, otherwise it is set to one.

Table 757. VERIFY_MAC Command

Register	Value	Data Direction
CSE_CMD	0x06	—
CSE_P1	Key ID	Input
CSE_P2	Message length (bits) address	Input
CSE_P3	Message start address	Input
CSE_P4	MAC address	Input
CSE_P5	MAC length (bits) / Verification Status	Input / Output

40.5.7 Load Key

The LOAD_KEY command updates a memory slot using parameters specified in *Table 758* according to the SHE memory slot update protocol (see *Section 40.4.3: Secure Storage*). The 128-bit M1 message contains the UID, Key ID and Authentication Key ID. The 256-bit M2 message contains the new security flags, counter and the key value all encrypted using a derived key generated from the Authentication Key. The 128-bit M3 message is a MAC generated over messages M1 and M2. The 256-bit M4 message is the concatenation of the UID, Key ID, Authorization Key ID and the encrypted counter value. The 128-bit M5 message is the MAC calculated over message M4.

Table 758. LOAD_KEY Command

Register	Value	Data Direction
CSE_CMD	0x07	—
CSE_P1	M1 address	Input
CSE_P2	M2 address	Input
CSE_P3	M3 address	Input
CSE_P4	M4 address	Output
CSE_P5	M5 address	Output

Note: For firmware versions up to and including 0x122:

After the CSE has executed the LOAD_KEY command the lock bits in the Data Flash Low/Mid address space block Locking register (LML) and secondary lock bits in the Secondary Low/mid address space block Locking register (SLL) will be set to 1 (locked) regardless of the setting before. This means that before application software can write to the Data Flash, the appropriate lock and secondary lock bits must be cleared again (the lock and secondary lock bits in the LML and SLL registers are not preserved through a CSE LOAD_KEY operation).

For firmware versions after and including 0x123:

The lock and secondary lock bits in the LML and SLL registers are preserved through a CSE LOAD_KEY operation.

The Firmware version is reported in register CSE_P1 after the INIT_CSE command or in CSE_P3 after the secure boot process (SECURE_BOOT command).

For firmware versions up to and including 0x123:

If write access is required to the Data Flash after the user keys have been updated, an SOC reset is required if more than 30 updates have been made since the last SOC reset, changes to BOOT_MAC and BOOT_MAC_KEY must be counted towards this total.

For firmware versions after and including 0x124:

If write access is required to the Data Flash after the user keys have been updated no SOC reset is required.

40.5.8 Load Plain Key

The LOAD_PLAIN_KEY command updates the RAM key memory slot with a 128-bit plaintext key with the parameter shown in *Table 759*.

Table 759. LOAD_PLAIN_KEY Command

Register	Value	Data Direction
CSE_CMD	0x08	—
CSE_P1	Key address	Input

40.5.9 Export RAM Key

The EXPORT_RAM_KEY command exports the RAM key data with the parameters in *Table 760*. Only keys loaded with the LOAD_PLAIN_KEY command may be exported. The output messages are compatible with the messages used for LOAD_KEY (see *Section 40.5.7: Load Key*for details).

Table 760. EXPORT_RAM_KEY Command

Register	Value	Data Direction
CSE_CMD	0x09	—
CSE_P1	M1 address	Output
CSE_P2	M2 address	Output

Table 760. EXPORT_RAM_KEY Command

Register	Value	Data Direction
CSE_P3	M3 address	Output
CSE_P4	M4 address	Output
CSE_P5	M5 address	Output

40.5.10 Initialize RNG

The INIT_RNG command initializes the internal PRNG state with a seed value generated by the TRNG and sets the CSE_SR[RIN] flag. It takes 1024 TRNG clock cycles to generate a seed value. The CSE_CR[DIV] field must be properly configured before this command is executed (see *Section 40.3.2: Register descriptions*). There is a very small probability that this command may return a TRNG error (EC=0x12) even when the TRNG is operating properly (see *Section 40.4.7: Random Number Generation*). The INIT_RNG command must be called after each reset before the CMD_RND command is issued. The command has no parameters as shown *Table 761*.

Table 761. INIT_RNG Command

Register	Value	Data Direction
CSE_CMD	0x0A	—

40.5.11 Extend PRNG Seed

The EXTEND_SEED command extends the state of the PRNG using a 128-bit entropy input value. The current PRNG state and the input data are compressed into a new PRNG state value. This command is issued with the parameter shown *Table 762*.

Table 762. EXTEND_SEED Command

Register	Value	Data Direction
CSE_CMD	0x0B	—
CSE_P1	Entropy value address	Input

40.5.12 Generate Random Number

The RND command generates a 128-bit random value and updates the state of the internal PRNG. The PRNG state must be initialized after reset using the INIT_RNG command before this command can be issued. This command is issued with the parameter shown in *Table 763*

Table 763. RND Command

Register	Value	Data Direction
CSE_CMD	0x0C	—
CSE_P1	Random value address	Output

40.5.13 Secure Boot

The SECURE_BOOT command loads the command processor firmware and memory slot data from the CSE Flash blocks, and then it executes the SHE secure boot protocol using the parameters shown in *Table 764* (see *Section 40.4.6: Secure Boot* for details). The bootloader size is a 32-bit value. This command is issued during the boot processes (in some device boot modes) by the BAM or system boot logic. The CSE firmware version is loaded into CSE_P3 register.

Table 764. SECURE_BOOT Command

Register	Value	Data Direction
CSE_CMD	0x0D	—
CSE_P1	Bootloader size (bytes)	Input
CSE_P2	Bootloader start address	Input
CSE_P3	Firmware version	Output

40.5.14 Boot Failure

The BOOT_FAILURE command sets the CSE_SR[BFN] flag and clears the CSE_SR[BOK] flag to disable use of memory slots that have the BOOT_PROT flag set. This command is issued with no parameters as shown in *Table 765*.

Table 765. BOOT_FAILURE Command

Register	Value	Data Direction
CSE_CMD	0x0E	—

40.5.15 Boot OK

The BOOT_OK command sets the CSE_SR[BFN] flag and leaves the CSE_SR[BOK] flag set to confirm successful completion of the secure boot processes. This enables the use of memory slots that have the BOOT_PROT flag set if they are not disabled for some other reason. This command is issued with no parameters as shown in *Table 766*.

Table 766. BOOT_OK Command

Register	Value	Data Direction
CSE_CMD	0x0F	—

40.5.16 Get ID

The GET_ID command returns the UID, CSE_SR[24:31] and a 128-bit MAC calculated over the concatenation of a 128-bit input challenge value, UID and CSE_SR[24:31]. The MASTER_ECU_KEY is used for the MAC calculation. A value of zero is returned for the MAC if the MASTER_ECU_KEY slot is empty. The UID output is a 128-bit value with the 8 least significant bits set to zero. This command is issued with the parameters shown in *Table 767*.

Table 767. GET_ID Command

Register	Value	Data Direction
CSE_CMD	0x10	—
CSE_P1	Challenge address	Input
CSE_P2	UID output address	Output
CSE_P3	CSE_SR[24:31] value	Output
CSE_P4	MAC address	Output

40.5.17 Cancel

The CANCEL command aborts processing of the current command and clears the CSE_SR[BSY] flag. This command is issued with no parameters as shown in *Table 768*.

Table 768. CANCEL Command

Register	Value	Data Direction
CSE_CMD	0x11	—

40.5.18 Debug Challenge

The DEBUG_CHAL command generates a 128-bit random challenge output value that is used in conjunction with the DEBUG_AUTH command. The PRNG state must be initialized after reset using the INIT_RNG command before this command can be issued. This command is issued with the parameter shown in *Table 769*.

Table 769. DEBUG_CHAL Command

Register	Value	Data Direction
CSE_CMD	0x12	—
CSE_P1	Challenge address	Output

40.5.19 Debug Authorization

The DEBUG_AUTH command erases all user keys and sets the CSE_SR[IDB] flag which enables internal debugging if the 128-bit authorization input value is valid and no memory slots are write protected. The authorization input is generated using the DEBUG_CHAL command output and the UID as described in the *SHE Functional Specification*. If the DEBUG_CHAL command is not issued before the DEBUG_AUTH command a command sequence error (EC=0x02) is returned. However, other commands may be issued between the DEBUG_CHAL and DEBUG_AUTH commands. This command is issued with the parameter shown in *Table 770*.

Table 770. DEBUG_AUTH Command

Register	Value	Data Direction
CSE_CMD	0x13	—
CSE_P1	Authorization address	Input

Note: For firmware versions up to and including 0x123:

If write access is required to the Data Flash after the user keys have been erased an SOC reset is required.

For firmware versions after and including 0x124:

If write access is required to the Data Flash after the user keys have been erased no SOC reset is required.

40.5.20 Generate TRNG Random Number

The TRNG_RND command generates a 128-bit random output value using the TRNG. It takes 1024 TRNG clock cycles to generate a random value. The CSE_CR[DIV] field must be properly configured before this command is executed (see *Section 40.3.2: Register descriptions*). This command takes much longer to execute than the RND command which should normally be used to generate random values. There is a very small probability that this command may return a TRNG error (EC=0x12) even when the TRNG is operating properly. This command is issued with the parameter shown in *Table 771*.

Table 771. TRNG_RND Command

Register	Value	Data Direction
CSE_CMD	0x14	—
CSE_P1	Random value address	Output

40.5.21 Initialize CSE

The INIT_CSE command loads the command processor firmware and memory slot data from the CSE Flash blocks into local memory. It does not execute the secure boot protocol. The CSE firmware version is loaded into the CSE_P1 register as shown in *Table 772*. This command must be issued before any other commands when using device boot modes that do not support secure boot.

Table 772. INIT_CSE Command

Register	Value	Data Direction
CSE_CMD	0x15	—
CSE_P1	Firmware version	Output

41 JTAG Controller (JTAGC)

41.1 Introduction

Figure 848 is a block diagram of the JTAG Controller (JTAGC) block.

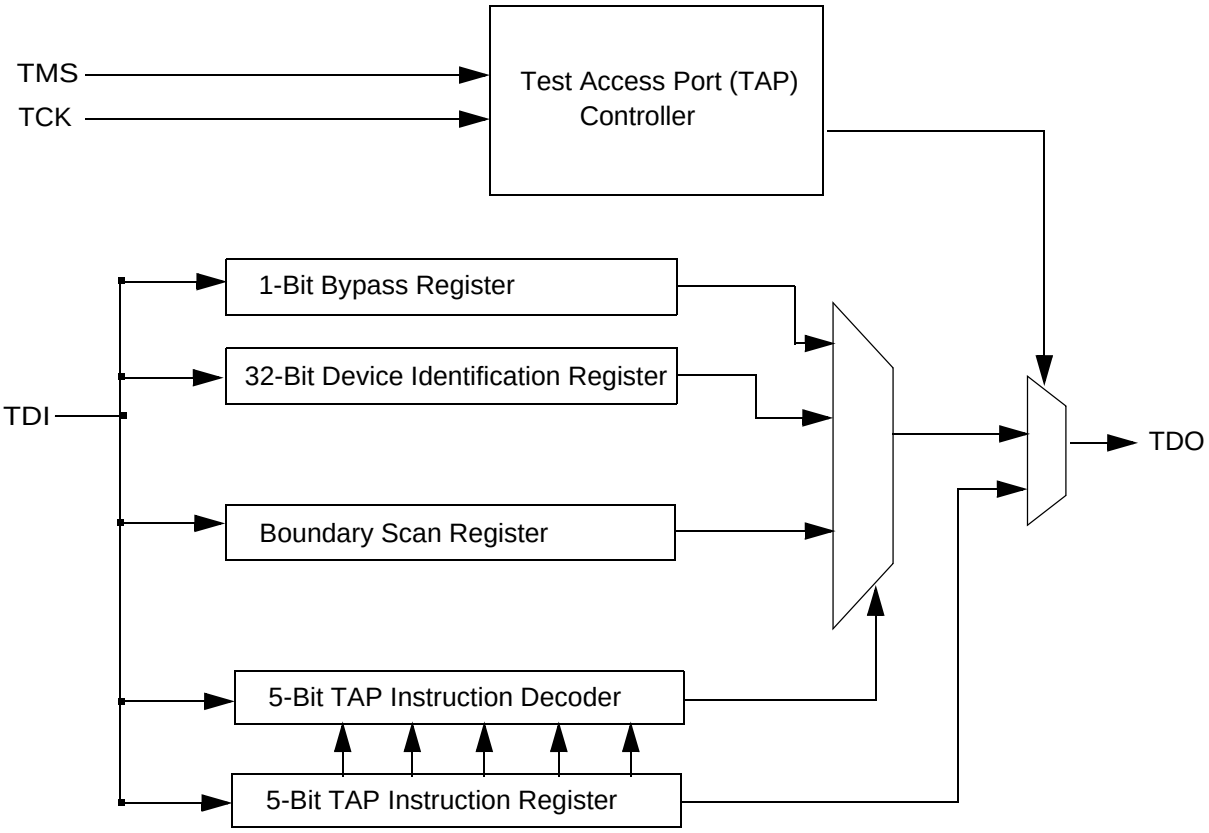


Figure 848. JTAG STL (IEEE 1149.1) block diagram

41.1.1 Overview

The JTAGC block 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. All data input to and output from the JTAGC block is communicated in serial format.

41.1.2 Features

The JTAGC block 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)
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions as well as several public and private device-specific instructions. Refer to *Table 774* for a list of supported instructions.
- Sharing of the TAP with other TAP controllers via ACCESS_AUX_TAP_x instructions.
- Test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry.

41.1.3 Modes of Operation

The JTAGC block 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.

41.1.3.1 Reset

The JTAGC block is placed in reset when either power-on reset is asserted or the TMS input is held high for enough consecutive rising edges of TCK to sequence the TAP controller state machine into the Test-Logic-Reset state. Holding TMS high for 5 consecutive rising edges of TCK guarantees entry into the Test-Logic-Reset state regardless of the current TAP controller state. 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

41.1.3.2 IEEE 1149.1-2001 Defined Test Modes

The JTAGC block supports several IEEE 1149.1-2001 defined test modes. A test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data register(s) that may operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in Section 41.4.4, JTAGC Block Instructions.

41.1.3.3 Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC block 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.

41.2 External signal description

41.2.1 Overview

The JTAGC consists of 5 signals that connect to off chip development tools and allow access to test support functions. The JTAGC signals are outlined in *Table 773*.

Table 773. JTAG Signal Properties

Name	I/O	Function	Reset State	Pull(1)
TCK	Input	Test Clock	—	Down
TDI	Input	Test Data In	—	Up
TDO	Output	Test Data Out	High Z(2)	-
TMS	Input	Test Mode Select	—	Up

1. The pull is not implemented in this block. Pullup/pulldown devices are implemented in the pads.
2. TDO output buffer enable is negated when the JTAGC is not in the Shift-IR or Shift-DR states. A weak pull may be implemented at the TDO pad for use when JTAGC is inactive.

41.2.2 Detailed signal descriptions

This section describes each of the signals listed in *Table 773* in more detail.

41.2.2.1 TCK - Test Clock Input

Test Clock Input (TCK) is an input pin used to synchronize the test logic and control register access through the TAP.

41.2.2.2 TDI - Test Data Input

Test Data Input (TDI) is an input pin that receives serial test instructions and data. TDI is sampled on the rising edge of TCK.

41.2.2.3 TDO - Test Data Output

Test Data Output (TDO) is an output pin that transmits serial output for test instructions and data. TDO is three-stateable and is actively driven only in the Shift-IR and Shift-DR states of the TAP controller state machine, which is described in Section 41.4.3, TAP Controller State Machine. The TDO output of this block is clocked on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

41.2.2.4 TMS - Test Mode Select

Test Mode Select (TMS) is an input pin used to sequence the IEEE 1149.1-2001 test control state machine. TMS is sampled on the rising edge of TCK.

41.3 Register definition

This section provides a detailed description of the JTAGC block 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.

41.3.1 Register Descriptions

The JTAGC block registers are described in this section.

41.3.1.1 Instruction Register

The JTAGC block uses a 5-bit instruction register as shown in *Table 849*. The instruction register allows instructions to be loaded into the block 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.

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset:	0	0	0	0	1

Figure 849. 5-Bit Instruction Register

41.3.1.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

41.3.1.3 Device Identification Register

The device identification register, shown in *Figure 850*, allows the revision number, part number, manufacturer, and design center responsible for the design of the part 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. The part revision number (PRN) and part identification number (PIN) fields are system plugs, and the manufacturer identity code (MIC) is a constant value assigned to the manufacturer by the JEDEC.

The shift register LSB is forced to logic 1 on the rising edge of TCK following entry into the Capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are forced to the value of the device identification register on the rising edge of TCK following entry into the Capture-DR state.

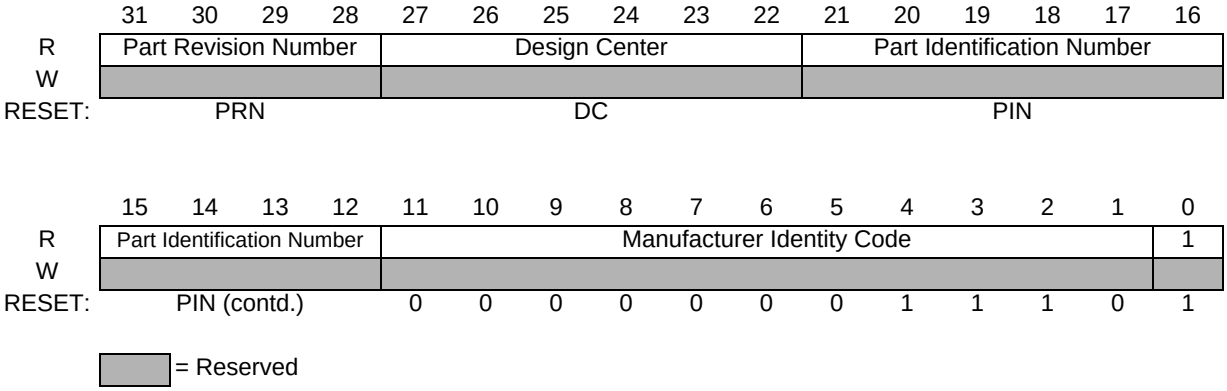


Figure 850. Device Identification Register

PRN — Part Revision Number
Bits [31:28] contain the revision number of the part.

DC — Bits [27:22] indicate the design center.

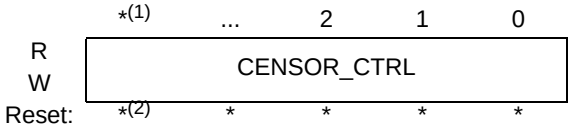
PIN — Part Identification Number
Bits [21:12] contain the part number of the device.

MIC — Manufacturer Identity Code
Bits [11:1] contain the reduced Joint Electron Device Engineering Council (JEDEC) ID.

Bit [0] — IDCODE Register ID
Bit [0] identifies this register as the device identification register and not the bypass register

41.3.1.4 CENSOR_CTRL Register

The CENSOR_CTRL register is a 64-bit shift register path from TDI to TDO selected when the ENABLE_CENSOR_CTRL instruction is active. The default reset value of the CENSOR_CTRL register is 64'b0. The CENSOR_CTRL register transfers its value to a parallel hold register on the rising edge of TCK when the TAP controller state machine is in the Update-DR state. Once the ENABLE_CENSOR_CTRL instruction is executed, the register value will remain valid until a JTAG reset occurs.



1. The size of CENSOR_CTRL is 64 bits.
2. The reset value of CENSOR_CTRL is 64'b0.

Figure 851. CENSOR_CTRL Register

CENSOR_CTRL — Censorship Control

The CENSOR_CTRL bits are used to control chiptop censorship functions.

41.3.1.5 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 41.4.5, Boundary Scan. The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file.

41.4 Functional Description

41.4.1 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.

41.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC block 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 41.4.4.8, 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 852*. This applies for the instruction register, test data registers, and the bypass register.

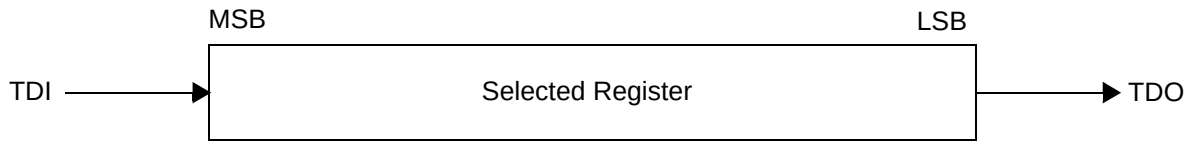
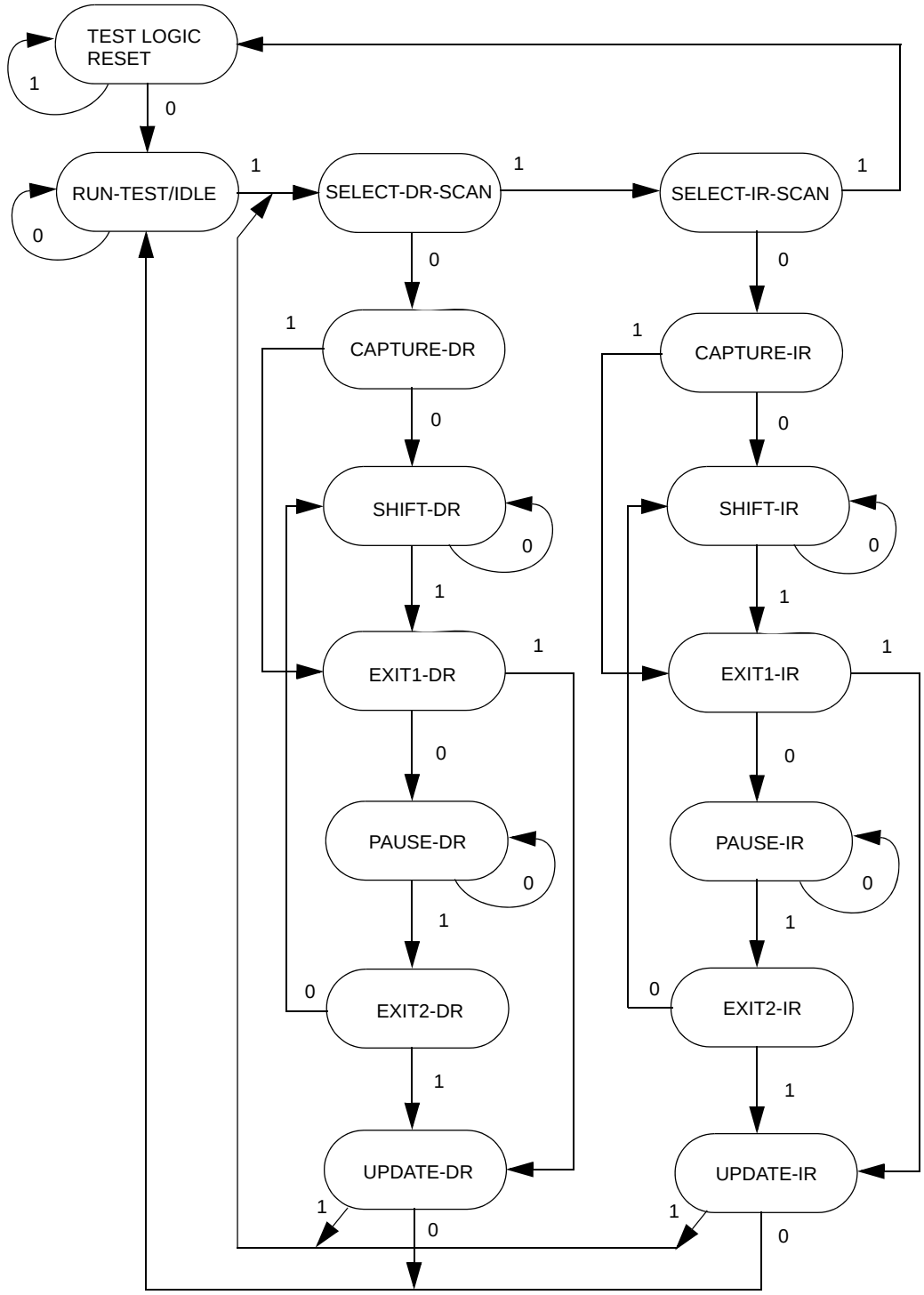


Figure 852. Shifting Data Through a Register

41.4.3 TAP Controller State Machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 853* 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 853* 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 853. IEEE 1149.1-2001 TAP Controller Finite State Machine

41.4.3.1 Selecting an IEEE 1149.1-2001 Register

Access to the JTAGC data registers is achieved by loading the instruction register with any of the JTAGC block 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 may be terminated once the required number of bits have been acquired.

41.4.4 JTAGC Block Instructions

The JTAGC block implements the IEEE 1149.1-2001 defined instructions listed in *Table 774*. This section gives an overview of each instruction; refer to the IEEE 1149.1-2001 standard for more details. All undefined opcodes are reserved.

Table 774. 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
ENABLE_CENSOR_CTRL	00111	Selects CENSOR_CTRL register
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_x(1)	10000-11110	Grants one of the auxiliary TAP controllers ownership of the TAP as shown in the cells below. The number of auxiliary TAP controllers sharing the port is SHARE_CNT
BYPASS	11111	Selects bypass register for data operations
Factory debug reserved	00101, 00110, 01010, 00111	Intended for factory debug only
Reserved(2)	All other opcodes	Decoded to select bypass register

1. The list of implemented tap codes are as follows:
   ACCESS_AUX_TAP_TCU = 11011
   ACCESS_AUX_TAP_ZO_NEXUS = 10010
   ACCESS_AUX_TAP_Z4_NEXUS = 10001
   ACCESS_AUX_TAP_NPC = 10000
   ACCESS_AUX_TAP_Z4_Z0_CASCADE_NEXUS = 10011
2. The manufacturer reserves the right to change the decoding of reserved instruction codes in the future

41.4.4.1 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 block is reset.

41.4.4.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- First, the SAMPLE portion of the 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/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.
- Secondly, the PRELOAD portion of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the Shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the Update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

41.4.4.3 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.

41.4.4.4 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.

41.4.4.5 ENABLE_CENSOR_CTRL Instruction

The ENABLE_CENSOR_CTRL instruction selects the CENSOR_CTRL register for connection as the shift path between TDI and TDO.

41.4.4.6 HIGHZ Instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active all output drivers are placed in an inactive drive state (e.g., high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

41.4.4.7 CLAMP Instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

41.4.4.8 ACCESS_AUX_TAP_x Instructions

The JTAGC is configurable to allow up to fifteen other TAP controllers on the device to share the port with it. This is done by providing ACCESS_AUX_TAP_x instructions for each of these TAP controllers. When this instruction is loaded, control of the JTAG pins are 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. Instructions not used to access an auxiliary TAP controller on a device are treated like the BYPASS instruction.

41.4.4.9 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.

41.4.5 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.

41.5 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 block 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

42 Nexus Development Interface (NDI)

42.1 Introduction

SPC564Bxx and SPC56ECxx contains multiple Nexus clients (Nexus3+) that communicate over an auxiliary port interface compliant to IEEE-ISTO 5001-2010 standard and a JTAG port compliant with IEEE 1149.1 standard.

Nexus3+ module provides real-time development capabilities for e200z0h and e200z4d core processor in compliance with Class 3 of the IEEE-ISTO 5001-2010 standard, with additional Class 4 features available.

The communication with NDI is handled via the auxiliary port and the JTAG port.

- The auxiliary port comprises 16 output pins and one input pin. The output pins include one message clock out (MCKO) pin, 12 message data out (MDO) pins, one message start/end out ($\overline{\text{MSEO}}$) pins, and one event out ($\overline{\text{EVTO}}$) pin. Event in ($\overline{\text{EVTI}}$) is the only input pin for the auxiliary port.
- The JTAG port 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. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC). See *Section 42.4: Memory Map and Registers*, for the JTAGC opcodes to access the different Nexus clients.

Note: Nexus3+ capabilities are only supported in the 256 BGA package.

42.2 Block diagram

Figure 854 shows a functional block diagram of the NDI. *Figure 855* shows an implementation block diagram of the NDI, which shows how the individual Nexus blocks are combined to form the NDI.

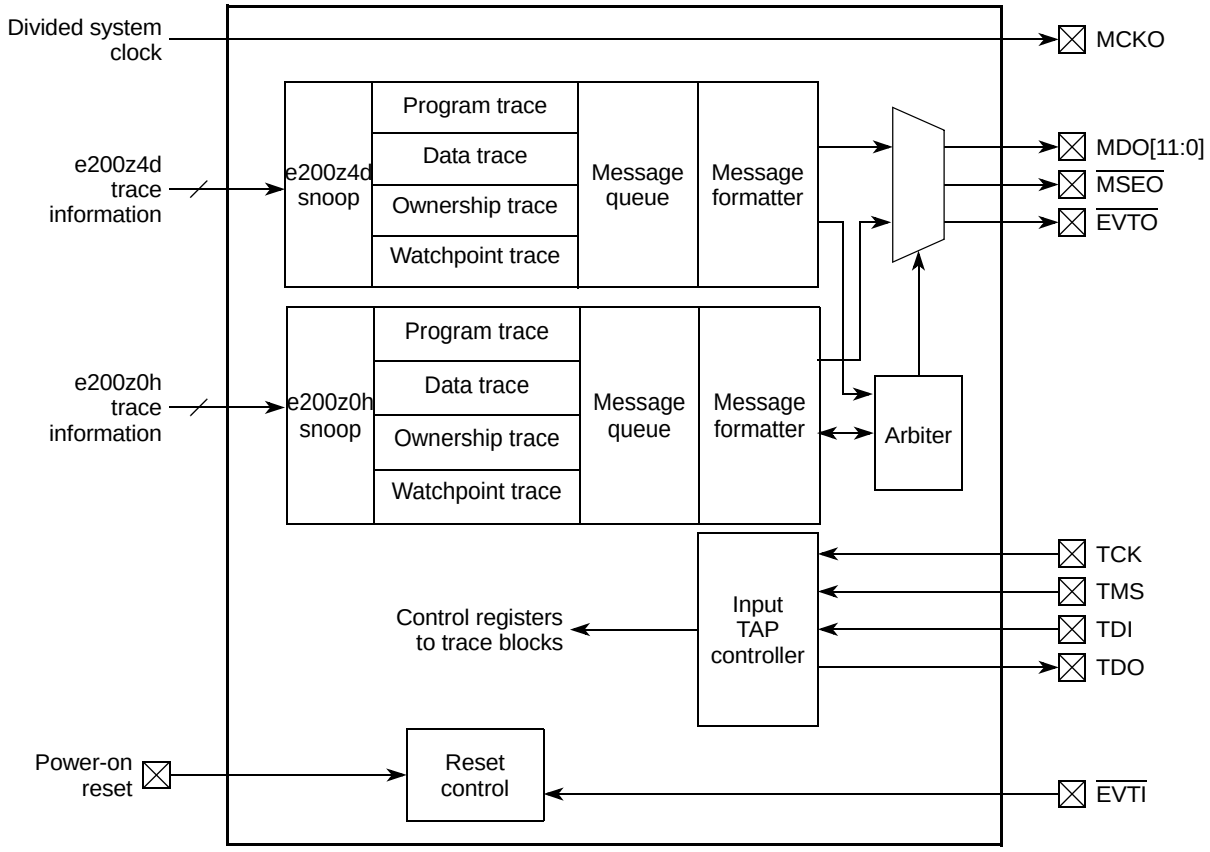


Figure 854. NDI functional block diagram

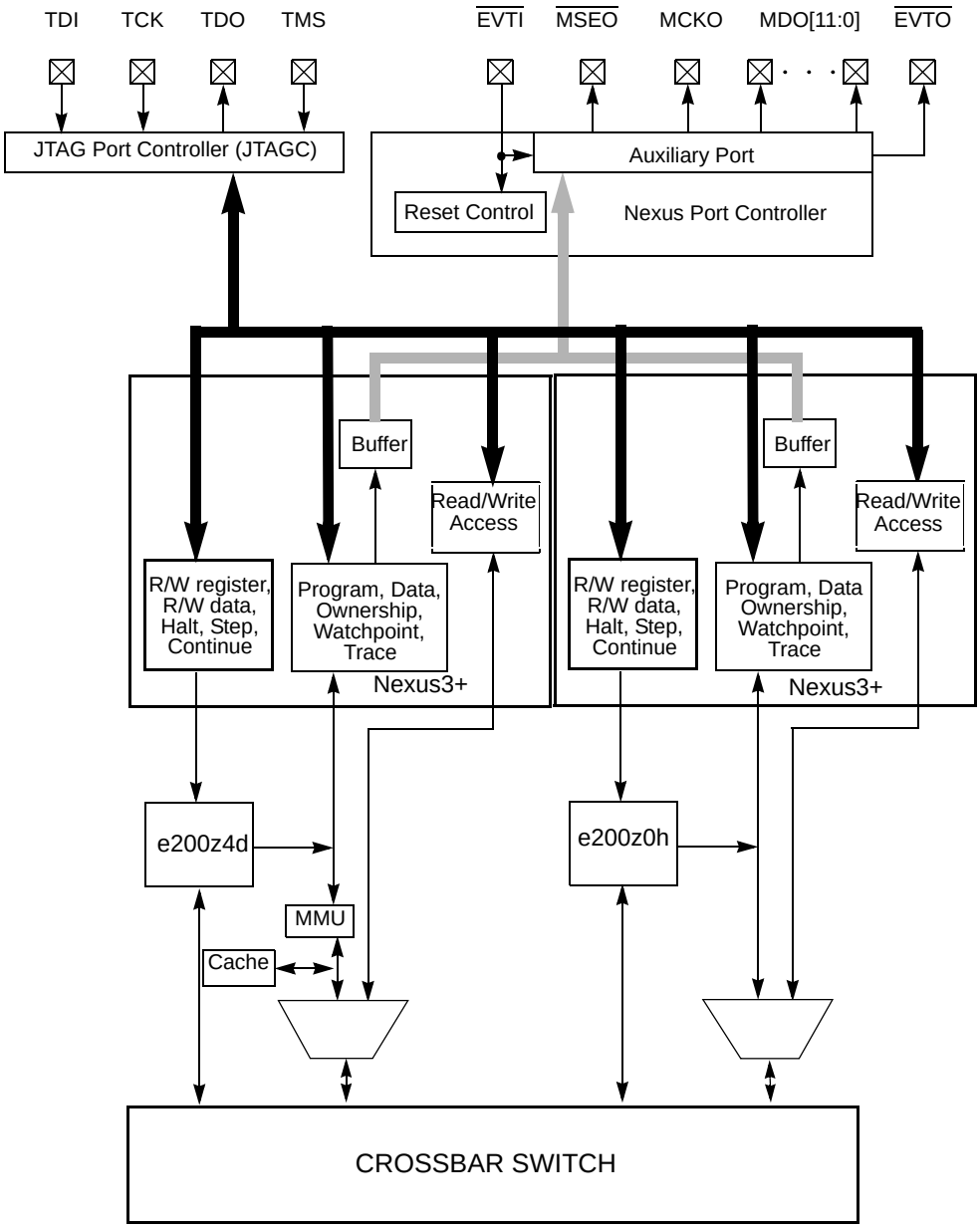


Figure 855. NDI Implementation block diagram

42.2.1 NDI Features

The NDI module of the SPC564Bxx and SPC56ECxx is compliant with the IEEE-ISTO 5001-2010 standard. The following features are implemented:

- 21-bit full duplex pin interface for high throughput, including existing four JTAG pins
  - Two modes are supported: full port mode (FPM) and reduced port mode (RPM). FPM comprises 12 MDO pins. RPM comprises eight MDO pins, and can be used to increase the number of GPIOs. Care must be taken as bandwidth will be limited.
  - Auxiliary output port
    - One MCKO (Message clock out) pin
    - 12 MDO (Message data out) pins
    - One $\overline{\text{MSEO}}$ (Message start/end out) pins
    - One $\overline{\text{EVTO}}$ (Event out) pin
  - Auxiliary input port uses one $\overline{\text{EVTI}}$ (Event in) pin
  - JTAG port uses four pins (TDI, TDO, TMS, and TCK)
- The NPC block performs the following functions:
  - Controls arbitration for ownership of the Nexus Auxiliary Output Port between e200z0h and e200z4d Nexus
  - Nexus Device Identification Register and Messaging
  - Generates MCKO enable and frequency division control signals
  - Controls sharing of EVTO between e200z0h and e200z4d cores.
  - Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle.
  - Control of the device-wide debug mode
  - Generates asynchronous reset signal for Nexus blocks
- Host processor (e200z4d) and secondary processor (e200z0h) development support features:
  - IEEE-ISTO 5001-2010 standard class 3+ compliant.
    - Program trace via Branch Trace Messaging (BTM), with the option of using Branch history messaging to enhance message throughput.
    - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
    - 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 development tools to trace ownership flow.
    - Run-time access to the on-chip memory map via the JTAG port.
    - Watchpoint messaging (WPM) via the auxiliary port. This allows a watchpoint to be set, which then sends a watchpoint message each time the watchpoint is hit. Unlike watchpoint triggering, WPM does not stop the core or start trace.
    - Registers for Program Trace, Data Trace, Ownership Trace and Watchpoint Trigger.
  - Additional class 4 features
    - Watchpoint trigger enable of program and/or data trace messaging. This is an extension of WPM to allow a watchpoint to stop or start program or data trace.

  - Processor overrun control
  - All features controllable and configurable via JTAG port.
- Cross triggering — The capability for an EVTO (event out) signal from either the e200z4d or e200z0h Nexus3+ to generate a debug request to the other core, thus allowing both cores to enter debug mode within a short period of each other.

Note: Because SPC564Bxx and SPC56ECxx implements multiple Nexus blocks, the configuration of the Message Data Out pins is controlled by the NPC.

42.2.2 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 JTAG controller (JTAGC) block.

The NPC transitions out of the reset state immediately following negation of power-on reset.

42.2.2.1 Nexus Reset Mode

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.

42.2.2.2 Full-Port Mode

In the full port mode (FPM), 12 MDO pins (MDO[0..11]) pins are used to transmit Nexus messages. All trace features are available by writing to the configuration registers via the JTAG port. FPM is entered by asserting the MCKO_EN and FPM bits in the *Port Configuration Register (PCR)*.

42.2.2.3 Reduced-Port Mode

In the reduced-port mode (RPM), the number of MDO pins is reduced from 12 to 8 (MDO[0..7]). All trace features are available by writing to the configuration registers via jtag. However, the FIFO overflow errors are more likely to occur in this mode, especially if the data trace is enabled. RPM is entered by asserting the MCKO_EN bit and negating the FPM bit in the *Port Configuration Register (PCR)*.

42.2.2.4 Disabled-Port Mode

In the disabled-port mode, auxiliary output pin port enable signals are negated, thereby disabling the message transmission. Any debug feature that generates messages cannot be used. The primary features available are class 1 features and read/write access to the registers. Class 1 features include the ability to trigger a breakpoint event indication through EVTO.

42.2.2.5 Halt Mode

Halt mode logic is implemented in the Nexus port controller (NPC). When a request is made to enter halt mode, the NDI block completes monitoring of any pending bus transaction, transmits all messages already queued, and acknowledges the halt 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 the NDI.

42.2.2.6 Multi mode Nexus Tap

In this mode, Nexus Debug interface of Z4 and Z0 gets connected in a daisy-chain fashion. This will allow both the core to be stopped simultaneously.

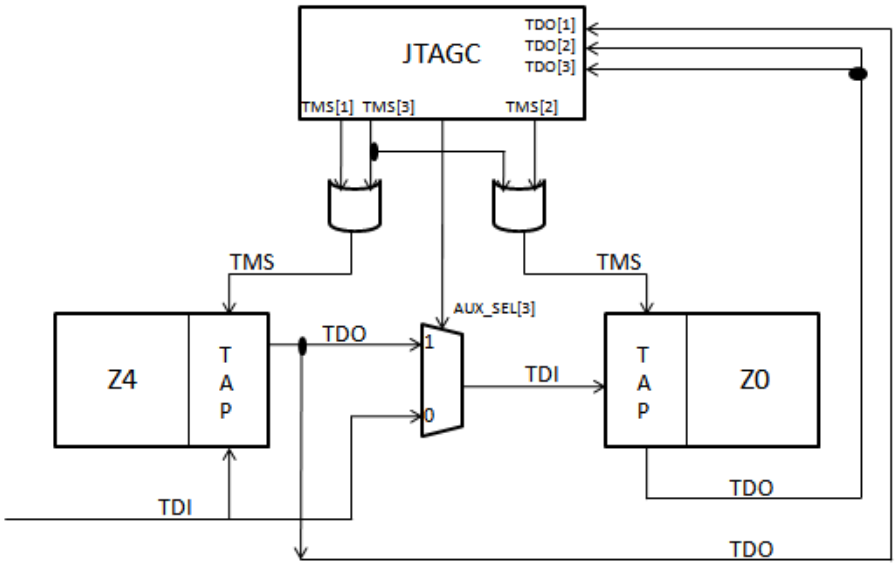


Figure 856. Multi mode Nexus Tap

42.3 External Signal Description

The auxiliary and JTAG pin interfaces provide for the transmission of messages from Nexus modules to the external development tools and for access to Nexus client registers. The auxiliary/JTAG pin definitions are outlined in *Table 775*.

Table 775. Signal Properties

Name	Port	Function
EVTO	Auxiliary	Event Out pin
EVTI	Auxiliary	Event In pin
MCKO	Auxiliary	Message Clock Out pin (from NPC)
MDO[11:0]	Auxiliary	Message Data Out pins
MSEO	Auxiliary	Message Start/End Out pins
TCK	JTAG	Test Clock Input

Table 775. Signal Properties (continued)

Name	Port	Function
TDI	JTAG	Test Data Input
TDO	JTAG	Test Data Output
TMS	JTAG	Test Mode Select Input

See *Chapter 4: Signal Description*, for detailed signal descriptions.

42.4 Memory Map and Registers

The NDI block contains no memory mapped registers. Nexus registers are accessed by the development tool via the JTAG port using a client select and a register index. The client select is controlled by loading the correct access instruction into the JTAG controller. After the desired client TAP is selected, OnCE registers for that client are accessible by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD). Nexus is enabled, and the associated Nexus registers become accessible, by loading the NEXUS_ENABLE instruction into the RS[0:6] field of the OCMD. When Nexus register access is enabled, the desired Nexus register is accessible using the index shown in *Table 776*.

Table 776. 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
e200z4d OnCE JTAG Instruction Opcodes		
NEXUS3_ACCESS	Opcode for e200z4d OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z4OnCE BYPASS instruction (10-bits)	0x7F
NEXUS_ACCESS	Opcode for e200z0h OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z0h OnCE BYPASS instruction (10-bits)	0x7F

Table 777. NDI Registers

Nexus Register	Nexus Access Opcode	Read/ Write	Read Address	Write Address
NPC Registers				
Device ID (DID)		R	0x00	0x05
Port Configuration Register (PCR)		R/W	0x7F	0x07F
e200z4d Control/Status Registers				
Development Control 1 (DC1)	0x2	R/W	0x04	0x05
Development Control 2 (DC2)	0x3	R/W	0x06	0x07

Table 777. NDI Registers

Nexus Register	Nexus Access Opcode	Read/ Write	Read Address	Write Address
Development Control 3 (DC3)	0x4	R/W	0x08	0x09
Development Control 4 (DC4)	0x5	R/W	0x0A	0x0B
Read/Write Access Control/Status (RWCS)	0x7	R/W	0x0E	0x0F
Read/Write Access Address (RWA)	0x9	R/W	0x12	0x13
Read/Write Access Data (RWD)	0xA	R/W	0x14	0x15
Watchpoint Trigger (WT)	0xB	R/W	0x16	0x17
Reserved	0xC	R/W	0x18	0x19
Data Trace Control (DTC)	0xD	R/W	0x1A	0x1B
Data Trace Start Address1 (DTSA1)	0xE	R/W	0x1C	0x1D
Data Trace Start Address2 (DTSA2)	0xF	R/W	0x1E	0x1F
Data Trace Start Address3 (DTSA3)	0x10	R/W	0x20	0x21
Data Trace Start Address4(DTSA4)	0x11	R/W	0x22	0x23
Data Trace End Address1 (DTEA1)	0x12	R/W	0x24	0x25
Data Trace End Address2 (DTEA2)	0x13	R/W	0x26	0x27
Data Trace End Address3 (DTEA3)	0x14	R/W	0x28	0x29
Data Trace End Address4 (DTEA4)	0x15	R/W	0x2A	0x2B
Development Status (DS)	0x30	R	0x60	-
Reserved	0x31	R/W	0x62	0x63
Overrun Control (OVCR)	0x32	R/W	0x64	0x65
Watchpoint Mask (WMSK)	0x33	R/W	0x66	0x67
Reserved	0x34	-	0x68	0x69
Program Trace Start Trigger Control (PTSTC)	0x35	R/W	0x6A	0x6B
Program Trace End Trigger Control (PTETC)	0x36	R/W	0x6C	0x6D
Data Trace Start Trigger Control (DTSTC)	0x37	R/W	0x6E	0x6F
Data Trace End Trigger Control (DTETC)	0x38	R/W	0x70	0x71
Reserved	0x39 -> 0x3F	-	0x72->0x7E	0x73->7F
e200z0h Control/Status Registers				
Development Control 1 (DC1)	0x2	R/W	0x04	0x05
Development Control 2 (DC2)	0x3	R/W	0x06	0x07
Development Status (DS)	0x4	R	0x08	—
Development Control 3 (DC3)	0x4	R/W	0x08	0x09
Development Control 4 (DC4)	0x5	R/W	0x0A	0x0B
Read/Write Access Control/Status (RWCS)	0x7	R/W	0x0E	0x0F
Read/Write Access Address (RWA)	0x9	R/W	0x12	0x13

Table 777. NDI Registers

Nexus Register	Nexus Access Opcode	Read/ Write	Read Address	Write Address
Read/Write Access Data (RWD)	0xA	R/W	0x14	0x15
Watchpoint Trigger (WT)	0xB	R/W	0x16	0x17
Reserved	0xC	R/W	0x18	0x19
Data Trace Control (DTC)	0xD	R/W	0x1A	0x1B
Data Trace Start Address1 (DTSA1)	0xE	R/W	0x1C	0x1D
Data Trace Start Address2 (DTSA2)	0xF	R/W	0x1E	0x1F
Reserved	0x10-> 0x11	-	0x20->0x22	0x21->23
Data Trace End Address1 (DTEA1)	0x12	R/W	0x24	0x25
Data Trace End Address2 (DTEA2)	0x13	R/W	0x26	0x27
Reserved	0x14 -> 0x2F	-	0x28->0x5E	0x29->5F
Development Status (DS)	0x30	R	0x60	-
Reserved	0x31	R/W	0x62	0x63
Overrun Control (OVCR)	0x32	R/W	0x64	0x65
Watchpoint Mask (WMSK)	0x33	R/W	0x66	0x67
Reserved	0x34	-	0x68	0x69
Program Trace Start Trigger Control (PTSTC)	0x35	R/W	0x6A	0x6B
Program Trace End Trigger Control (PTETC)	0x36	R/W	0x6C	0x6D
Data Trace Start Trigger Control (DTSTC)	0x37	R/W	0x6E	0x6F
Data Trace End Trigger Control (DTETC)	0x38	R/W	0x70	0x71
Reserved	0x39 -> 0x3F	-	0x72->0x7E	0x73->7F

42.4.1 NDI Functional Description

The NDI block is implemented by integrating the following blocks:

- Nexus Port Controller Block
- Nexus e200z4d Development Interface (OnCE and Nexus3+ subblocks)
- Nexus e200z0h Development Interface (OnCE and Nexus3+ subblocks)
- NPC_HNDSHK module

Note: The TAP controller logic, reset logic, and some miscellaneous logic are duplicated in all these blocks.

42.4.1.1 Enabling Nexus Clients for TAP Access

Once the NDI is 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. Once the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

42.4.1.2 TAP Sharing

Each of the individual Nexus blocks on the MCU implements a TAP controller for accessing its registers. The JTAGC controls the ownership of the TAP so that the interface to all of these individual TAP controllers appears to be a single port from outside the device. Once a Nexus client has been granted ownership of the TAP, 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.

42.4.1.3 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of power-on 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 the FPM bit selects full-port mode.

When writing to the PCR, the PCR LSB must be written to a logic 0. Setting the LSB of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 778 describes the NDI configuration options.

Table 778. 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

42.4.1.4 Programmable MCKO frequency

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include SYS_CLK, one-half, one-quarter, and one-eighth SYS_CLK speed.

Table 779 shows the MCKO_DIV encodings. In this table, SYS_CLK represents the system clock frequency.

Table 779. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
0b000(1)	SYS_CLK
0b001	SYS_CLK/2
0b010	Reserved
0b011	SYS_CLK/4
0b100	Reserved
0b101	Reserved

Table 779. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
0b110	Reserved
0b111	SYS_CLK/8

1. The SYS_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins (60 MHz).

Note: The Nexus Port Controller Port Configuration Register MCKO Divider bits (NPC_PCR[MCKO_DIV]) can be set to 0b000 to select a 1X clock rate as the Nexus Auxiliary output port frequency for the MCKO and MDO pins. Depending on the system frequency, this may force the MCKO and MDO pins to switch at a frequency higher than can be supported by the pins. This maximum frequency is specified in the device electrical specification of the Nexus MCKO and MDO pins. Insure that the maximum operating frequency of the MDO and MCKO pins is not violated when setting the NPC_PCR[MCKO_DIV] values.

Note: For the mode where Z0:Z4 frequency ratio is 1:2, MCKO Divide value of 1 is not supported.

42.4.1.5 Cross Triggering Control

To enable a debug event in one core to cause a debug event in the other core at approximately the same time, the $\overline{\text{EVTO}}$ signal from the e200z0h Nexus3+ or e200z4d Nexus3+ is connected to the other core's devt2 input. When enabled in each core's Nexus1 DBCR0 register, a pulse of the devt2 signal causes a debug event to occur. In this case, only one external $\overline{\text{EVTO}}$ signal is generated and each core controls whether or not $\overline{\text{EVTO}}$ causes a debug event to occur.

Interconnection of debug mode control signals are shown in *Figure 857*.

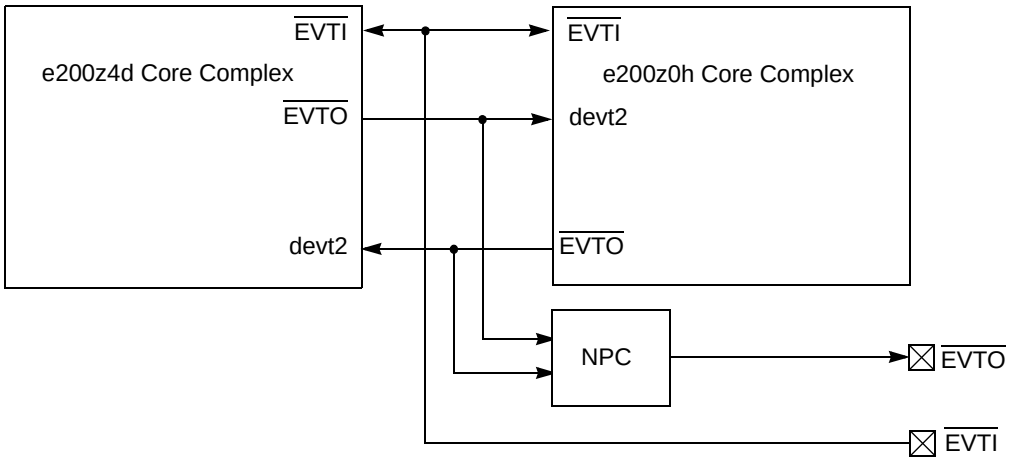


Figure 857. Debug Mode Control Interconnections

42.5 Nexus Port Controller (NPC)

The Nexus Port Controller (NPC) is that part of the NDI that controls access and arbitration of the device's internal Nexus modules. The NPC contains the port configuration register (PCR) and the device identification register (DID).

42.5.1 Introduction

Figure 858 is a block diagram of the Nexus Port Controller (NPC) block.

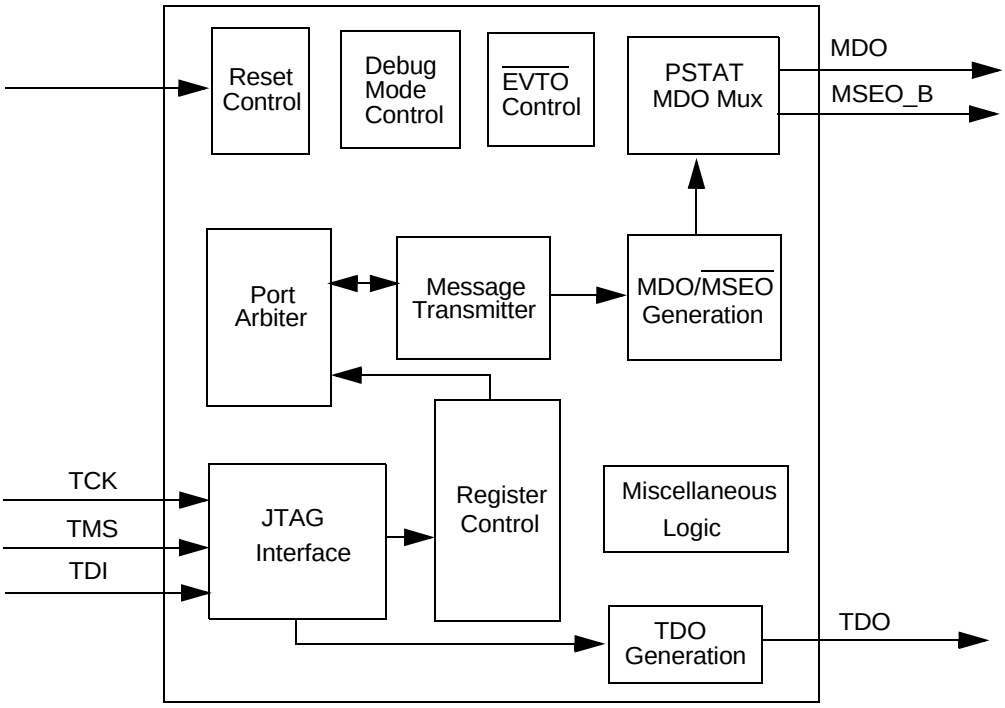


Figure 858. Nexus Port Controller Block Diagram

42.5.2 NPC features

The NPC performs the following functions:

- Controls arbitration for ownership of the Nexus Auxiliary Output Port
- Nexus Device Identification Register and Messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$
- Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle
- Control of the device-wide debug mode
- Generates asynchronous reset signal for Nexus blocks based on power-on reset status

42.5.3 NPC memory map

Table 780 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass register (refer to *Section 42.5.4.1: Bypass Register*) and instruction register (refer to *Section 42.5.4.2: Instruction Register*) have no index values. These registers are not accessed in the same manner as Nexus client registers.

Table 780. NPC Memory Map

Index	Register Name	Register Description	Size (bits)
0	DID	Device ID register	32
127	PCR	Port configuration register	32

42.5.4 NPC Register descriptions

This section consists of NPC register descriptions.

42.5.4.1 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the CAPTURE-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

42.5.4.2 Instruction Register

The NPC uses a 4-bit instruction register as shown in *Figure 859*. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the module to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

As shown in *Section 41.4.3: TAP Controller State Machine*, instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

	3	2	1	0
R	Previous Instruction Opcode			
W	Instruction Opcode			
Reset:	BYPASS Instruction Opcode (0xF)			

Figure 859. 4-Bit Instruction Register

42.5.4.3 Nexus Device ID Register (DID)

The NPC device identification register, shown in *Figure 860*, 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. See *Section 42.5.5.5.1: NPC IEEE 1149.1-2001 (JTAG) TAP*. This register is read-only.

Reg Index: 0x00 Access: User read only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PRN				DC				PIN							
W																
Reset (1)	PRN				DC				PIN							

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIN (*continued*)				MIC											1
W																
Reset	PIN (cont'd.)				0	0	0	0	0	0	0	1	1	1	0	1

Figure 860. Nexus Device ID Register (DID)

1. Part Revision Number default value is 0x0 for the device's initial mask set and changes for each mask set revision.

Table 781. DID field descriptions

Field	Description
PRN	Part Revision Number. Contains the revision number of the part. This field changes with each revision of the device or module.
DC	Design Center. Indicates the Freescale design center. This value is 0x2B.
PIN	Part Identification Number. The value is 0x249.
MIC	Manufacturer Identity Code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID.
bit 0	Fixed Per JTAG 1149.1. Always set to 1.

42.5.4.4 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 and SLEEP_SYNC bits, but must preserve the original state of the remaining bits in the register.

Note: The mode (MCKO_GT) or clock division (MCKO_DIV) bits must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	0	NEXCFG	0	0	0	ALC_NUM_LN		0	0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	LP_DBG	0	0	0	0	0	LP2_SYN	LP1_SYN	0	0	0	0	0	0	0	PSTAT_EN
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Reserved

Figure 861. Port Configuration Register (PCR)

Table 782. PCR field descriptions

Name	Description
FPM	Full Port Mode The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 = 8 MDO pins are used to transmit messages 1 = 12 MDO pins are used to transmit messages
MCKO_GT	MCKO Clock Gating Control This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 0 = MCKO gating is disabled 1 = MCKO gating is enabled
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. MCKO clock is driven to zero MCKO clock is enabled
MCKO_DIV	MCKO Division Factor The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted.
EVT_EN	EVTO/EVTI Enable This bit enables the EVTO/EVTI port functions0 = EVTO/EVTI port disabled 1= EVTO/EVTI port enabled
NEXCFG	Nexus Configuration Select Generic Nexus control bit. 0 = NEXCFG cleared 1= NEXCFG set
LP_DBG_EN(1)	Low Power Debug Enable This bit enables debug functionality on exit from low power modes on supported devices. 0 = Low power debug disabled 1= Low power debug enabled

Table 782. PCR field descriptions

Name	Description
LPn_SYN[1]	Low Power Mode n Synchronization These bits are used to synchronize the entry into low power modes between the device and debug tool. Supported devices set these bits before a pending entry into low power mode. After reading the bit as set, the debug tool then clears the bit to acknowledge to the device that it may enter the low power mode. 0 = Low power mode entry acknowledged 1= Low power mode entry pending
PSTAT_EN	Processor Status Mode Enable(2) This bit enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. 0 = PSTAT mode disabled 1= PSTAT mode enabled

1. The TCK frequency must be lower than system frequency during normal operation and during low power operation.
2. PSTAT Mode is intended for factory processor debug only. The PSTAT_EN bit should be written to disable PSTAT mode if Nexus messaging is desired. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled.

42.5.5 NPC Functional Description

42.5.5.1 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 HALT0, STOP0, or STANDBY0 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 HALT0 or STOP0 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 STANDBY0 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.

42.5.5.2 NPC Reset Configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field.

42.5.5.3 Auxiliary Output Port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the individual modules and arbitrates for access to the port.

42.5.5.3.1 Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the $\overline{\text{MSEO}}$ functions. The $\overline{\text{MSEO}}$ pin is used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

Figure 862 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

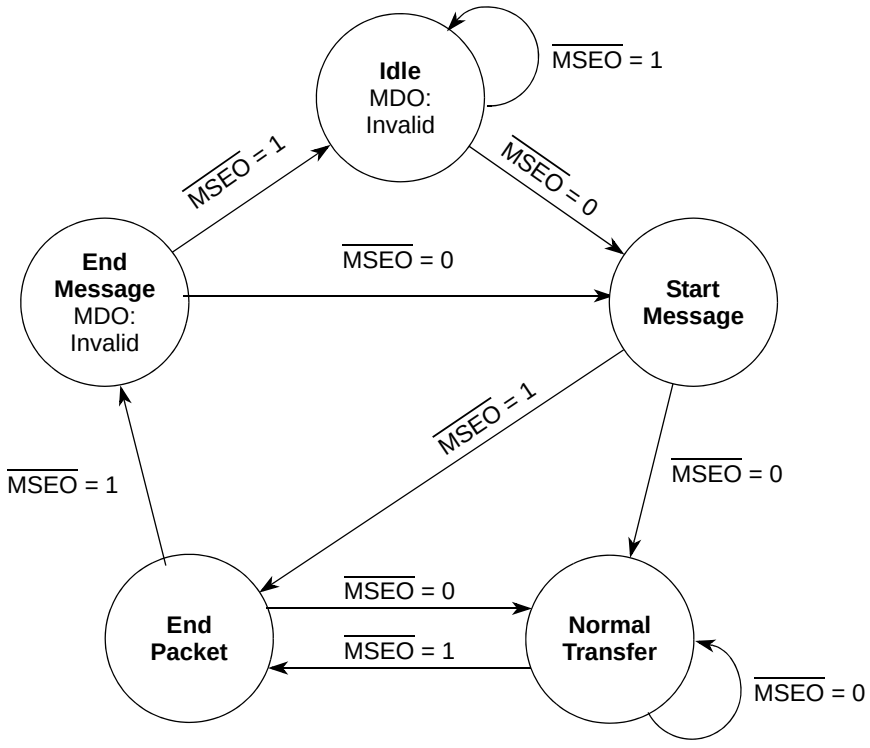


Figure 862. $\overline{\text{MSEO}}$ Transfers

42.5.5.3.2 Output Messages

42.5.5.4 Output Messages

In addition to sending out messages generated in other Nexus blocks, the NPC can also output the device ID message contained in the device ID register and the port replacement output message on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 783 describes the device ID and port replacement output messages that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 783. NPC Output Messages

Message Name	Min. Packet Size (bits)	Max Packet Size (bits)	Packet Type	Packet Name	Packet Description
Device ID Message	6	6	fixed	TCODE	Value = 1
	32	32	fixed	ID	DID register contents

Figure 863 shows the various message formats that the pin interface formatter has to encounter. Note that for variable-length fields, the transmitted size of the field is determined from the range of the least significant bit to the most significant non-zero-valued bit (i.e. most significant zero-valued bits are not transmitted).

Figure 863. Message field sizes

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size(1)(bits)	Max Size(2) (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

1. Minimum information size. The actual number of bits transmitted depends on the number of MDO pins
2. Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

The double edges in *Figure 863* indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

42.5.5.4.0.1 Rules of Messages

42.5.5.5 Rules of Message

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field may start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets may start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. *Figure 864* shows the transmission sequence of a message that is made up of a TCODE followed by two fields.

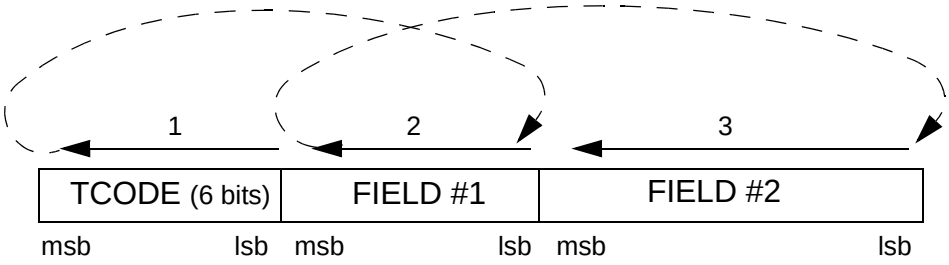


Figure 864. Transmission Sequence of Messages

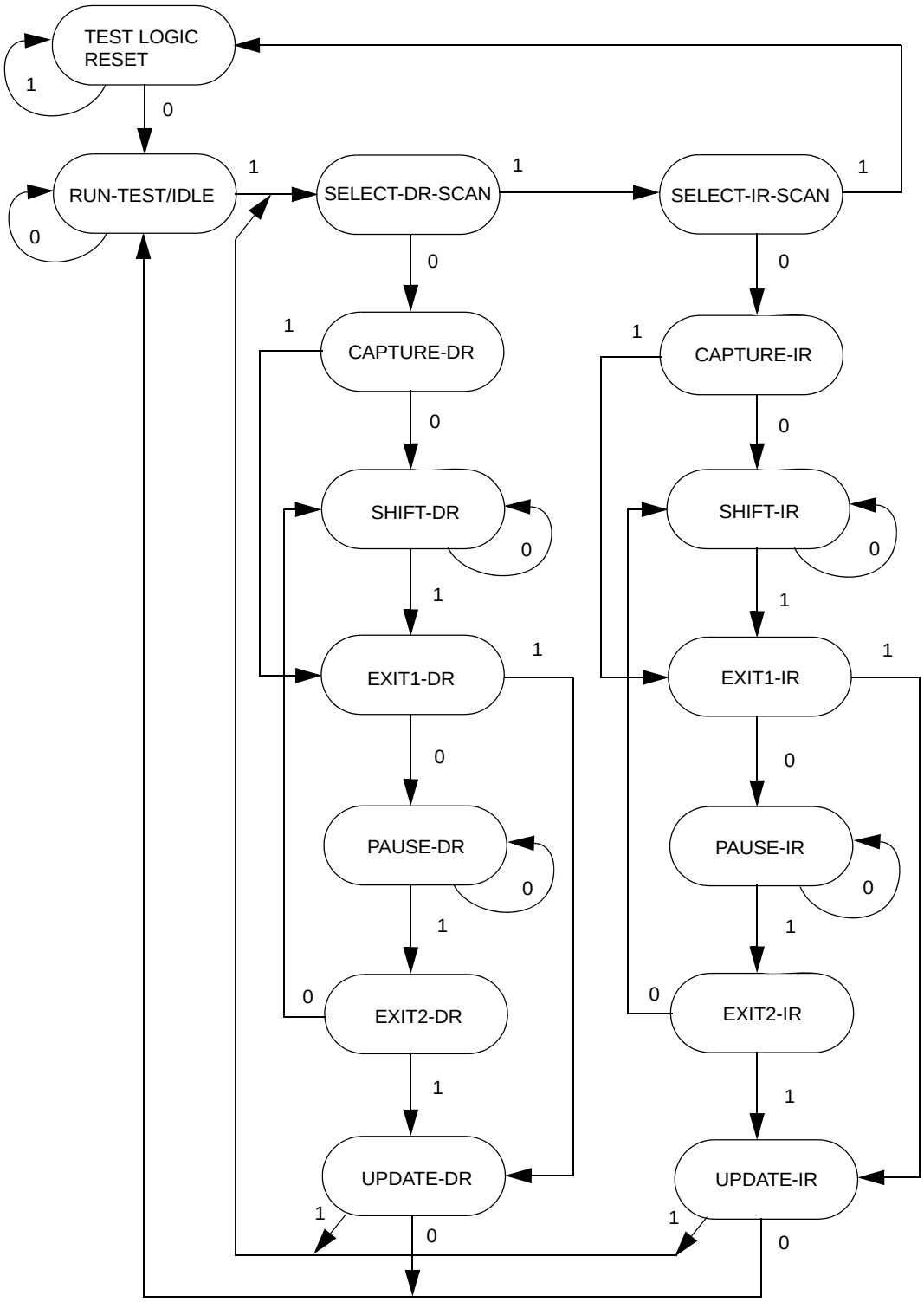
42.5.5.5.1 NPC IEEE 1149.1-2001 (JTAG) TAP

The NPC uses the IEEE 1149.1-2001 TAP, which uses the state machine for accessing registers. The NPC also implements the Nexus controller state machine as defined by the IEEE-ISTO 5001-2010 standard as shown in *Figure 866*.

The instructions implemented by the NPC TAP controller are listed in *Table 776*.

42.5.5.5.1.1 Enabling the NPC TAP Controller

Assertion of the power-on reset signal resets the NPC TAP controller. When not in power-on reset, the NPC TAP controller is enabled by loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC. Loading the NEXUS-ENABLE instruction then grants access to NPC registers.



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 865. IEEE 1149.1-2001 TAP Controller State Machine

42.5.5.5.1.2 Retrieving Device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction {npc_basic}. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the Nexus Device ID register through the TAP.{npc_basic} Transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR, if the NPC is enabled.{npc_basic} Transmission of the device identification message serially via TDO is achieved by performing a read of the register contents as described in *Section 42.5.5.5.1.4: Selecting a Nexus Client Register*.{npc_basic}

42.5.5.5.1.3 Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled by loading the NPC NEXUS-ENABLE instruction when NPC has ownership of the TAP. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in *Figure 866*, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. *Table* illustrates the IEEE® 1149.1 sequence to load the NEXUS-ENABLE instruction.

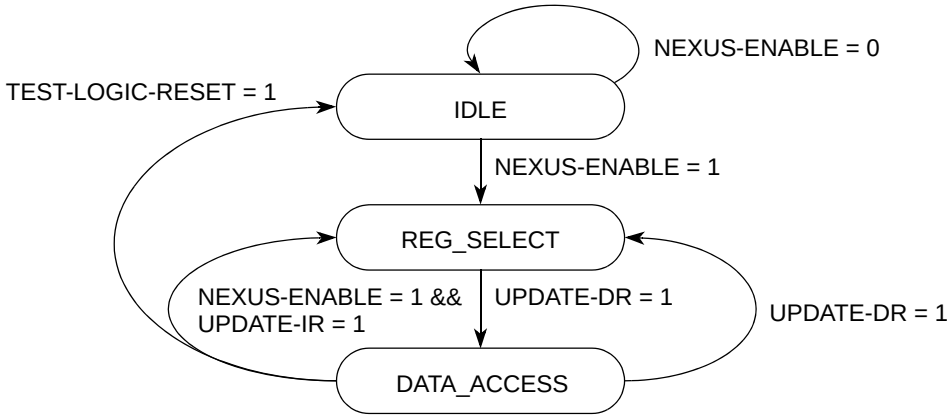


Figure 866. NEXUS Controller State Machine

Table 784. Loading NEXUS-ENABLE instruction

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	IDLE	Transitional state
2	1	SELECT-IR-SCAN	IDLE	Transitional state
3	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKS				
12	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
13	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
14	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

42.5.5.5.1.4 Selecting a Nexus Client Register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path. The Nexus Controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit register address index, as illustrated in *Figure 867*. The read/write control bit is set to 1 for writes and 0 for reads.

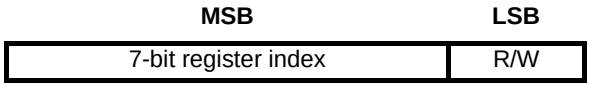


Figure 867. IEEE 1149.1 Controller Command Input

The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (LSB first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

Table 785 illustrates a sequence which writes a 32-bit value to a register

Table 785. Write to a 32-Bit Nexus Client Register

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
12	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
13	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
14	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
15	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
16	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
48	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.
49	1	UPDATE-DR	DATA_ACCESS	Value written to register
50	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

42.5.5.5.2 Nexus JTAG Port Sharing

Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers. Only the module whose ACCESS_AUX_TAP instruction is loaded

has control of the TAP. This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. Once a Nexus module has ownership of the TAP, that module acts like a single-bit shift register, or bypass register, if no register is selected as the shift path.

42.5.5.5.3 MCKO

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV[2:0] field in the PCR. Possible operating frequencies include full, one-half, one-quarter, and one-eighth SYS_CLK speed. MCKO is enabled by setting the MCKO_EN bit in the PCR.

42.5.5.5.4 $\overline{\text{EVTO}}$ Sharing

The NPC controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. $\overline{\text{EVTO}}$ is driven for one MCKO period whenever any module drives its $\overline{\text{EVTO}}$. The sharing mechanism is a logical AND of all incoming $\overline{\text{EVTO}}$ signals from Nexus blocks, thereby asserting $\overline{\text{EVTO}}$ whenever any block drives its $\overline{\text{EVTO}}$. The order these signals are connected at the NPC input does not matter. When no MCKO is active, such as in disabled mode, the NPC assumes an MCKO frequency of one-half system clock speed when driving $\overline{\text{EVTO}}$. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

42.5.5.5.5 Nexus Reset Control

The power-on-reset signal is used as the primary reset signal for the NPC, which is also used by the NPC to generate a single-bit reset signal for other Nexus blocks.

42.5.6 NPC Initialization/Application Information

To initialize the TAP for NPC register accesses, the following sequence is required:

1. Enable the NPC TAP controller. This is achieved by loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC.
2. Load the TAP controller with the NEXUS-ENABLE instruction.

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE®-ISTO 5001-2010 standard for more detail.

42.6 Nexus3+ Module

The e200z0h and e200z4d Nexus3+ module provides real-time development capabilities for the core processors in compliance with the IEEE-ISTO Nexus 5001-2010 standard.

The module provides development support capabilities without requiring address and data pins, thus providing visibility of internal instructions and data access.

A portion of the pin interface (the JTAG port) is also shared with the OnCE / Nexus 1 unit. The IEEE-ISTO 5001-2010 standard defines an extensible auxiliary port which is used in conjunction with the JTAG port in core processors.

The Nexus modules are coupled to the CPU core and monitor a variety of signals including addresses, data, control signals, status signals, etc.

42.6.1 Introduction

This section defines the auxiliary pin functions, transfer protocols, and standard development features of the Nexus3+ module. The development features supported are Program Trace, Data Trace, Watchpoint Messaging, Ownership Trace, Data Acquisition Messaging, and Read/Write Access via the JTAG interface. The Nexus3+ module also supports two Class 4 features: Watchpoint Triggering, and Processor Overrun Control.

42.6.2 Block Diagram

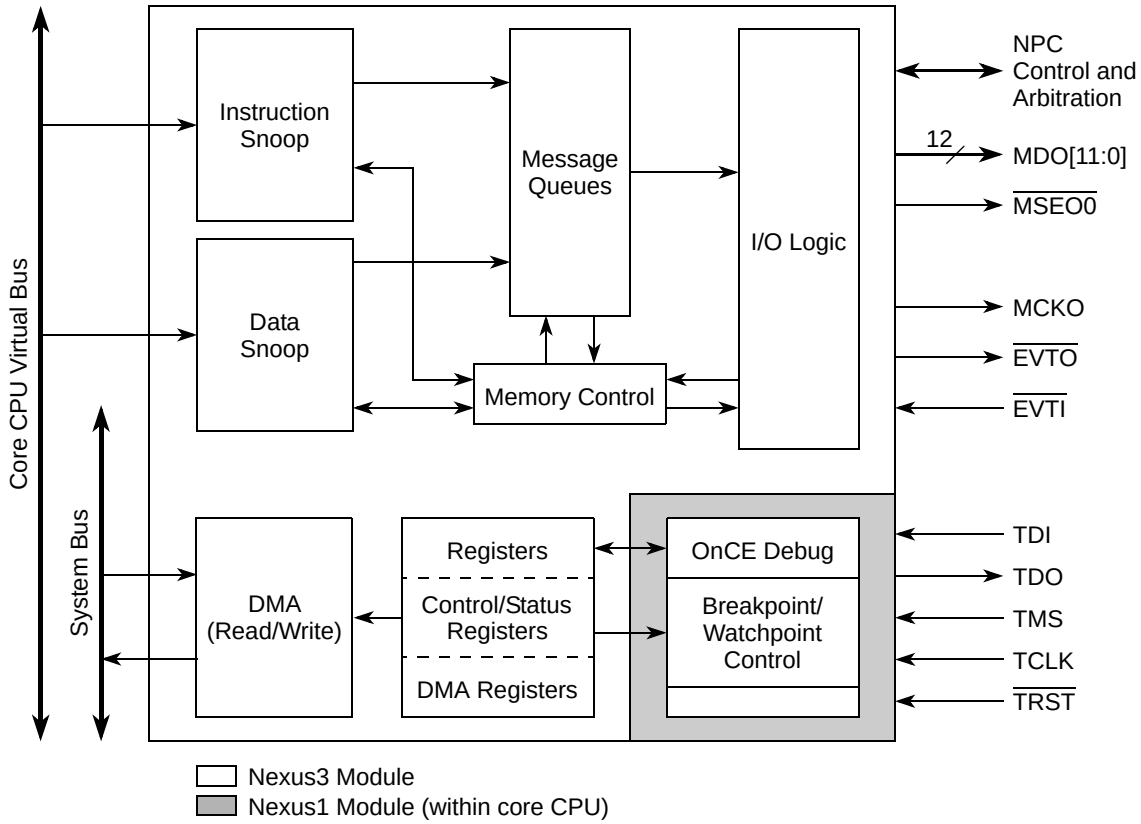


Figure 868. Nexus3 Functional Block Diagram

42.6.3 Overview

Table 786 contains a set of terms and definitions associated with the Nexus3 module.

Table 786. Terms and Definitions

Term	Description
IEEE-ISTO 5001	Consortium & standard for real-time embedded system design. World wide Web documentation at http://www.ieee-isto.org/Nexus5001
Auxiliary Port	Refers to Nexus auxiliary port. Used as auxiliary port to the IEEE 1149.1 JTAG interface.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions, and the number of sequential instructions executed between each taken branch.
Data Read Message (DRM)	External visibility of data reads to memory-mapped resources.
Data Write Message (DWM)	External visibility of data writes to memory-mapped resources.
Data Trace Messaging (DTM)	External visibility of how data flows through the embedded system. This may include DRM and/or DWM.
Data Acquisition Messaging (DQM)	Data Acquisition Messaging (DQM) allows code to be instrumented to export customized information to the Nexus Auxiliary Output Port.
JTAG Compliant	Device complying to IEEE 1149.1 JTAG standard
JTAG IR & DR Sequence	JTAG Instruction Register (IR) scan to load an opcode value for selecting a development register. The JTAG IR corresponds to the OnCE command register (OCMD). The selected development register is then accessed via a JTAG Data Register (DR) scan.
Nexus1	The core (OnCE) debug module. This module integrated with each Zen processor provides all static (core halted) debug functionality. This module is compliant with Class1 of the IEEE-ISTO 5001 standard.
Ownership Trace Message (OTM)	Visibility of process/function that is currently executing.
Public Messages	Messages on the auxiliary pins for accomplishing common visibility and controllability requirements
SoC	“System-on-a-Chip”. SoC signifies all of the modules on a single die. This generally includes one or more processors with associated peripherals, interfaces & memory modules.
Standard	The phrase “according to the standard” is used to indicate according to the IEEE-ISTO 5001 standard.
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
Watchpoint	A Data or Instruction Breakpoint or other debug event which does not cause the processor to halt. Instead, a pin is used to signal that the condition occurred. A Watchpoint Message may also be generated.

42.6.4 Enabling Nexus3 Operation

The Nexus3 module is enabled by loading a single instruction into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (refer to *Table 776*). For the Nexus3 module, the OCMD value

is 0b00_0111_1100. Once enabled, the module is ready to accept control input via the JTAG pins. See *Section 42.4.1.1: Enabling Nexus Clients for TAP Access* for more information.

Enabling the Nexus3 module automatically enables the generation of Debug Status Messages.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by cycling through the state machine using the TMS pin. The Nexus module also is disabled if a power-on-reset (POR) event occurs. If the Nexus3+ module is disabled, no trace output is provided, and the module disables (drives inactive) auxiliary port output pins MDO[11:0], MSEO, and MCKO. Nexus registers are not available for reads or writes.

42.6.5 TCODEs Supported

The Nexus3 pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2010 standard defines a set of public messages. The Nexus3 module supports the public TCODEs seen in *Table 787*. Each message contains multiple packets transmitted in the order shown in the table.

Table 787. Public TCODEs supported

Message Name	Minimum Field Size (bits)	Maximum Field Size (bits)	Field Name	Field Type	Packet Description
Debug Status	6	6	TCODE	fixed	TCODE number = 0
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	8	8	STATUS	fixed	Debug Status Register (DS[31:24])
Ownership Trace Message	6	6	TCODE	fixed	TCODE number = 2
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	12	PROCESS	variable	Task/Process ID tag
Program Trace - Direct Branch Message	6	6	TCODE	fixed	TCODE number = 3
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	ICNT	variable	# sequential instructions executed since last taken branch
Program Trace - Indirect Branch Message	6	6	TCODE	fixed	TCODE number = 4
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (IS) indicator
	1	8	ICNT	variable	# sequential instructions completed since last predicate instruction, transmitted instruction count, or taken change of flow
	1	32	U-ADDR	variable	unique part of target address for taken branches/exceptions

Table 787. Public TCODEs supported

Message Name	Minimum Field Size (bits)	Maximum Field Size (bits)	Field Name	Field Type	Packet Description
Data Trace - Data Write Message	6	6	TCODE	fixed	TCODE number = 5
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (DS) indicator
	4	4	DSZ	fixed	data size
	1	32	U-ADDR	variable	unique portion of the data write address
	1	32	DATA	variable	data write value(s)
Data Trace - Data Read Message	6	6	TCODE	fixed	TCODE number = 6
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (DS) indicator
	4	4	DSZ	fixed	data size (Refer to *Table 792*)
	1	32	U-ADDR	variable	unique portion of the data read address
	1	32	DATA	variable	data read value(s)
Data Acquisition Message	6	6	TCODE	fixed	TCODE number = 7
	4	4	SRC	fixed	source processor identifier
	8	8	DQTAG	fixed	identification tag taken from $DEVENT_{DQTAG}$ register field
	1	32	DQDATA	variable	exported data taken from DDAM register
Error Message	6	6	TCODE	fixed	TCODE number = 8
	4	4	SRC	fixed	source processor identifier
	4	4	ETYPE	fixed	error type
	8	8	ECODE	fixed	error code
Program Trace - Direct Branch Message w/ Sync	6	6	TCODE	fixed	TCODE number = 11
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	ICNT	variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	variable	full target address (leading zeros truncated)
Program Trace - Indirect Branch Message w/ Sync	6	6	TCODE	fixed	TCODE number = 12
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (IS) indicator
	1	8	ICNT	variable	# sequential instructions completed since last predicate instruction, transmitted instruction count, or taken change of flow
	1	32	F-ADDR	variable	full target address (leading zeros truncated)

Table 787. Public TCODEs supported

Message Name	Minimum Field Size (bits)	Maximum Field Size (bits)	Field Name	Field Type	Packet Description
Data Trace - Data Write Message w/ Sync	6	6	TCODE	fixed	TCODE number = 13
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (DS) indicator
	4	4	DSZ	fixed	data size (Refer to *Table 792*)
	1	32	F-ADDR	variable	full access address (leading zeros truncated)
	1	32	DATA	variable	data write value(s) (see Data Trace section for details)
Data Trace - Data Read Message w/ Sync	6	6	TCODE	fixed	TCODE number = 14
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (DS) indicator
	4	4	DSZ	fixed	data size (Refer to *Table 792*)
	1	32	F-ADDR	variable	full access address (leading zeros truncated)
	1	32	DATA	variable	data read value(s) (see Data Trace section for details)
Watchpoint Message	6	6	TCODE	fixed	TCODE number = 15
	4	4	SRC	fixed	source processor identifier
	1	32	WPHIT	variable	Field indicating watchpoint source(s) (leading zeros truncated)
Resource Full Message	6	6	TCODE	fixed	TCODE number = 27
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	4	4	RCODE	fixed	resource code (Refer to *Table 790*) - indicates which resource is the cause of this message
	1	32	RDATA	variable	branch / predicate instruction history (see Section)
Program Trace - Indirect Branch History Message	6	6	TCODE	fixed	TCODE number = 28 (see Note below)
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (IS) indicator
	1	8	I-CNT	variable	# sequential instructions completed since last predicate instruction, transmitted instruction count, or taken change of flow
	1	32	U-ADDR	variable	unique part of target address for taken branches/exceptions
	1	32	HIST	variable	branch / predicate instruction history

Table 787. Public TCODEs supported

Message Name	Minimum Field Size (bits)	Maximum Field Size (bits)	Field Name	Field Type	Packet Description
Program Trace - Indirect Branch History Message w/ Sync	6	6	TCODE	fixed	TCODE number = 29 (see Note below)
	4	4	SRC	fixed	source processor identifier
	1	1	MAP	fixed	Address Space (IS) indicator
	1	8	I-CNT	variable	# sequential instructions completed since last predicate instruction, transmitted instruction count, or taken change of flow
	1	32	F-ADDR	variable	full target address (leading zero (0) truncated)
	1	32	HIST	variable	branch / predicate instruction history
Program Trace - Program Correlation Message	6	6	TCODE	fixed	TCODE number = 33
	4	4	SRC	fixed	source processor identifier
	4	4	EVCODE	fixed	event correlated w/ program flow (Refer to *Table 791*)
	2	2	CDF	fixed	# fields of information in CDATA. 01 - one field (CDATA1), 10 - two fields (CDATA1 + CDATA2), 11 - three fields (CDATA1 + CDATA2 + CDATA3)
	1	8	I-CNT	variable	# sequential instructions completed since last predicate instruction, transmitted instruction count, or taken change of flow
	1	32	CDATA1	variable	correlation data field 1 - [branch / predicate instruction
	0	32	CDATA2	variable	correlation data field 2- PID/IS info or TLB info (F-ADDR_V for virtual address or tlbivax EA)
	0	32	CDATA3	variable	correlation data field 3 - TLB info -ADDR_P for physical address

Note: Program Trace can be implemented using either Branch History/Predicate Instruction Messages, or traditional Direct/Indirect Branch Messages. The user can select between the two types of Program Trace. If the Branch History method is selected, the shaded TCODES above will not be messaged out.

Table 788 shows the error code encodings used when reporting an error via the Nexus3 Error Message.

Table 788. Error Code Encoding (TCODE = 8)

Error Code	Description
xxxxxxx1	Watchpoint Trace Message(s) Lost
xxxxxx1x	Data Trace Message(s) Lost

Table 788. Error Code Encoding (TCODE = 8)

Error Code	Description
xxxxx1xx	Program Trace Message(s) Lost
xxxx1xxx	Ownership Trace Message(s) Lost
xxx1xxxx	Status Message(s) Lost (Debug Status messages, etc.)
xx1xxxxx	Data Acquisition Message(s) Lost
x1xxxxxx	Reserved
1xxxxxxx	Reserved

Table 789 shows the error type encodings used when reporting an error via the Nexus3 Error Message.

Table 789. Error Type Encoding (TCODE = 8)

Error Type	Description
0000	Message Queue Overrun caused one or more messages to be lost
0001	Contention with higher priority messages caused one or more messages to be lost
0010	Reserved
0011	Read/write access error
0100	Reserved
0101	Invalid access opcode (Nexus Register unimplemented)
0110 - 1111	Reserved

Table 790 shows the encodings used for resource codes for certain messages.

Table 790. RCODE values (TCODE = 27)

Resource Code	Description
0000	Program Trace Instruction counter reached 255 and was reset.
0001	Program Trace, Branch / Predicate Instruction History full. This type of packet is terminated by a stop bit set to 1 after the last history bit.

Table 791 shows the event code encodings used for certain messages.

Table 791. Event Code Encoding (TCODE = 33)

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only)
0010-0011	Reserved for future functionality
0100	Disabling Program Trace

Table 791. Event Code Encoding (TCODE = 33)

Event Code	Description
0101-1001	Reserved for future functionality
1010	Branch and link occurrence (direct branch function call)(1)
1011	New Address Translation established in the TLB
1100	Address Translation entries invalidated in the TLB
1101	Reserved for future functionality
1110	End of BookE tracing (trace disable or entry into a VLE page from a non-VLE page)
1111	End of VLE tracing (trace disabled or entry into a non-VLE page from a VLE page)

1. Only used for Program Trace - History Mode

Table 792 shows the data trace size encodings used for certain messages.

Table 792. Data Trace Size Encodings (TCODE = 5,6,13,14)

DTM Size Encoding	Transfer Size
0000	0 - no data
0001	Byte
0010	Halfword (2 bytes)
0011	Reserved
0100	Word (4 bytes)
0100-1111	Reserved

42.6.6 Memory Map

This section describes the Nexus3 programmer's model. Nexus3 registers are accessed using the JTAG/OnCE port in compliance with IEEE® 1149.1. See *Section 42.6.8: Register Access via JTAG / OnCE* for details on Nexus3 register access.

Table 793 details the register map for the Nexus3 module.

Table 793. Nexus3 Registers

Nexus Register	Nexus Access Opcode	Read/ Write	Read Address	Write Address
Client Select Control (CSC)(1)	0x1	R	0x02	-
Port Configuration Register (PCR)(1)	PCR_INDEX(2)	R/W	-	-
Development Control 1 (DC1)	0x2	R/W	0x04	0x05
Development Control 2 (DC2)	0x3	R/W	0x06	0x07

Table 793. Nexus3 Registers

Nexus Register	Nexus Access Opcode	Read/ Write	Read Address	Write Address
Development Control 3 (DC3)	0x4	R/W	0x08	0x09
Development Control 4 (DC4)	0x5	R/W	0x0A	0x0B
Read/Write Access Control/Status (RWCS)	0x7	R/W	0x0E	0x0F
Read/Write Access Address (RWA)	0x9	R/W	0x12	0x13
Read/Write Access Data (RWD)	0xA	R/W	0x14	0x15
Watchpoint Trigger (WT)	0xB	R/W	0x16	0x17
Reserved	0xC	R/W	0x18	0x19
Data Trace Control (DTC)	0xD	R/W	0x1A	0x1B
Data Trace Start Address1 (DTSA1)	0xE	R/W	0x1C	0x1D
Data Trace Start Address2 (DTSA2)	0xF	R/W	0x1E	0x1F
Reserved	0x10-> 0x11	-	0x20->0x22	0x21->23
Data Trace End Address1 (DTEA1)	0x12	R/W	0x24	0x25
Data Trace End Address2 (DTEA2)	0x13	R/W	0x26	0x27
Reserved	0x14 -> 0x2F	-	0x28->0x5E	0x29->5F
Development Status (DS)	0x30	R	0x60	-
Reserved	0x31	R/W	0x62	0x63
Overrun Control (OVCR)	0x32	R/W	0x64	0x65
Watchpoint Mask (WMSK)	0x33	R/W	0x66	0x67
Reserved	0x34	-	0x68	0x69
Program Trace Start Trigger Control (PTSTC)	0x35	R/W	0x6A	0x6B
Program Trace End Trigger Control (PTETC)	0x36	R/W	0x6C	0x6D
Data Trace Start Trigger Control (DTSTC)	0x37	R/W	0x6E	0x6F
Data Trace End Trigger Control (DTETC)	0x38	R/W	0x70	0x71
Reserved	0x39 -> 0x3F	-	0x72->0x7E	0x73->7F

1. The CSC and PCR registers are shown in this table as part of the Nexus programmer's model.
2. The "PCR_INDEX" is a parameter determined by the SoC.

42.6.7 Register Definition

42.6.7.1 Development Control Register 1 (DC1)

The development control registers are used to control the basic development features of the Nexus module. *Figure 869* shows DC1 and *Table 794* describes the register's fields.

Nexus Reg: 0x02 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC	MCK_DIV		0	PTM	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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	POTD	TSEN		EOC		EIC		0	0	TM					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 869. Development Control Register 1 (DC1)

Table 794. DC1 Field Descriptions

Field	Description
OPC	Output Port Mode Control. 0 Reduced-port mode configuration (8 MDO pins). 1 Full-port mode configuration (12 MDO pins).
MCK_DIV	MCK_DIV - MCKO Clock Divide Ratio (see note below) 00 - nex_mcko is 1x processor clock freq. 01 - nex_mcko is 1/2x processor clock freq. 10 - nex_mcko is 1/4x processor clock freq. 11 - nex_mcko is 1/8x processor clock freq.
PTM	Program Trace Method. 0 Program trace uses traditional branch messages. 1 Program trace uses branch history messages.
POTD	Periodic Ownership Trace Disable 0 Periodic Ownership Trace message events are enabled 1 Periodic Ownership Trace message events are disabled
TSEN	Timestamp Enable - (not implemented, write to 00) 00 Timestamp is disabled
EOC	$\overline{\text{EVTO}}$ Control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoints (configured in DC2). 01 $\overline{\text{EVTO}}$ upon entry into debug mode. 1X Reserved.
EIC	$\overline{\text{EVTI}}$ Control. 00 $\overline{\text{EVTI}}$ is used for synchronization (program trace/ data trace). 01 $\overline{\text{EVTI}}$ is used for debug request. 1X Reserved.
TM	Trace Mode(1). Any or all of the TM bits may set, enabling one or more traces. 000000 All Trace Disabled XXXXX1 Ownership Trace enabled XXXX1X Data Trace enabled XXX1XX Program Trace enabled XX1XXX Watchpoint Trace enabled X1XXXX Reserved 1XXXXX Data Acquisition Trace enabled

1. This field may be updated by hardware in response to watchpoint triggering. Writes to this field take precedence over hardware updates in the event of a collision.

Note: The Output Port Mode Control bit (OPC) and MCKO Clock Divide Ratio bits (MCK_DIV) MUST ONLY be modified during system reset or debug mode to insure correct output port and output clock functionality. It is also recommended that all other bits of the DC1 also only be modified in one of these two modes.

42.6.7.2 Development Control Register 2 (DC2)

DC2 is shown in *Figure 870* and its fields are described in *Table 795*.

Nexus Reg: 0x3 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	WEVTO[2]C				WEVTO[1]C				WEVTO[0]C			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 870. Development Control Register 2 (DC2)

Table 795. DC2 Field Descriptions

Field	Description
WEVTO[2]C	WEVTO[2]C- Watchpoint Event Out 2 Configuration 0000 No Watchpoints #0—14 trigger 0001 Watchpoint #0 triggers 0010 Watchpoint #1 triggers 0011 Watchpoint #2 triggers 0100 Watchpoint #3 triggers 0101 Watchpoint #4 triggers 0110 Watchpoint #5 triggers 0111 Watchpoint #6 triggers 1000 Watchpoint #7 triggers 1001 Watchpoint #8 triggers 1010 Watchpoint #9 triggers 1011 Watchpoint #10 triggers 1100 Watchpoint #11 triggers 1101 Watchpoint #12 triggers 1110 Watchpoint #13 triggers 1111 Watchpoint #14 triggers

Table 795. DC2 Field Descriptions (continued)

Field	Description
WEVTO[1]C	WEVTO[1]C- Watchpoint Event Out 1 Configuration 0000 No Watchpoints #0—14 trigger 0001 Watchpoint #0 triggers 0010 Watchpoint #1 triggers 0011 Watchpoint #2 triggers 0100 Watchpoint #3 triggers 0101 Watchpoint #4 triggers 0110 Watchpoint #5 triggers 0111 Watchpoint #6 triggers 1000 Watchpoint #7 triggers 1001 Watchpoint #8 triggers 1010 Watchpoint #9 triggers 1011 Watchpoint #10 triggers 1100 Watchpoint #11 triggers 1101 Watchpoint #12 triggers 1110 Watchpoint #13 triggers 1111 Watchpoint #14 triggers
WEVTO[0]C	WEVTO[0]C- Watchpoint Event Out 0 Configuration 0000 No Watchpoints #0—14 trigger 0001 Watchpoint #0 triggers 0010 Watchpoint #1 triggers 0011 Watchpoint #2 triggers 0100 Watchpoint #3 triggers 0101 Watchpoint #4 triggers 0110 Watchpoint #5 triggers 0111 Watchpoint #6 triggers 1000 Watchpoint #7 triggers 1001 Watchpoint #8 triggers 1010 Watchpoint #9 triggers 1011 Watchpoint #10 triggers 1100 Watchpoint #11 triggers 1101 Watchpoint #12 triggers 1110 Watchpoint #13 triggers 1111 Watchpoint #14 triggers

Table 795. DC2 Field Descriptions (continued)

Field	Description
EWC	$\overline{\text{EVTO}}$ Watchpoint Configuration. Any or all of the bits in EWC may be set to configure the $\overline{\text{EVTO}}$ watchpoint. 0000000000000000 No Watchpoints XXXXXXXXXXXXXXX1 Watchpoint XXXXXXXXXXXXXX1X Watchpoint XXXXXXXXXXXXX1XX Watchpoint XXXXXXXXXXXX1XXX Watchpoint XXXXXXXXXXX1XXXX Watchpoint XXXXXXXXXX1XXXXX Watchpoint XXXXXXXXX1XXXXXX Watchpoint XXXXXXXX1XXXXXXX Watchpoint XXXXXXX1XXXXXXXX Watchpoint XXXXXX1XXXXXXXXX Watchpoint XXXXX1XXXXXXXXXX Watchpoint XXXX1XXXXXXXXXXX Watchpoint XXX1XXXXXXXXXXXX Watchpoint XX1XXXXXXXXXXXXX Watchpoint X1XXXXXXXXXXXXXX Watchpoint 1XXXXXXXXXXXXXXX Watchpoint

Note: The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

42.6.7.3 Development Control Register 3 (DC3)

DC3 is shown in *Figure 871*. Its fields are described in *Table 796*.

Nexus Reg: 0x4 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	WEVTO[2]C				WEVTO[1]C				WEVTO[0]C			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 871. Development Control Register 3 (DC3)

Table 796. DC3 Field Descriptions

Field	Description
WEVTO[2]C	WEVTO[2]C- Watchpoint Event Out 2 Configuration 000 No Watchpoints #0—14 trigger 0001 Watchpoint #15 triggers 0010 Watchpoint #16 triggers 0011 Watchpoint #17 triggers 0100 Watchpoint #18 triggers 0101 Watchpoint #19 triggers 0110 Watchpoint #20 triggers 0111 Watchpoint #21 triggers 1000–1111 = Reserved
WEVTO[1]C	WEVTO[1]C- Watchpoint Event Out 1 Configuration
WEVTO[0]C	WEVTO[0]C- Watchpoint Event Out 0 Configuration
EWC	$\overline{\text{EVTO}}$ Watchpoint Configuration. 000000 No Watchpoints #16-#21 trigger XXXXX1 Watchpoint #16 triggers XXXX1X Watchpoint #17 triggers XXX1XX Watchpoint #18 triggers XX1XXX Watchpoint #19 triggers X1XXXX Watchpoint #20 triggers 1XXXXX Watchpoint #21 triggers

42.6.7.4 Development Control Register 4 (DC4)

DC4 is shown in *Figure 872*. Its fields are described in *Table 873*.

Nexus Reg: 0x4 Access: User read/write

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

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

Figure 872. Development Control Register 4 (DC4)

Table 797. DC4 Field Descriptions

Field	Description
EVCDM	Event Code (EVCODE) Mask1 0000000000000000 - No EVCODEs masked for Program Correlation Messages XXXXXXXXXXXXXXX1 - EVCODE #0 is masked for Program Correlation Messages XXXXXXXXXXXXXX1X - EVCODE #1 is masked for Program Correlation Messages XXXXXXXXXXXXX1XX - EVCODE #2 is masked for Program Correlation Messages XXXXXXXXXXXX1XXX - EVCODE #3 is masked for Program Correlation Messages XXXXXXXXXXX1XXXX - EVCODE #4 is masked for Program Correlation Messages XXXXXXXXXX1XXXXX - EVCODE #5 is masked for Program Correlation Messages XXXXXXXXX1XXXXXX - EVCODE #6 is masked for Program Correlation Messages XXXXXXXX1XXXXXXX - EVCODE #7 is masked for Program Correlation Messages XXXXXXX1XXXXXXXX - EVCODE #8 is masked for Program Correlation Messages XXXXXX1XXXXXXXXX - EVCODE #9 is masked for Program Correlation Messages XXXXX1XXXXXXXXXX - EVCODE #10 is masked for Program Correlation Messages XXXX1XXXXXXXXXXX - EVCODE #11 is masked for Program Correlation Messages XXX1XXXXXXXXXXXX - EVCODE #12 is masked for Program Correlation Messages XX1XXXXXXXXXXXXX - EVCODE #13 is masked for Program Correlation Messages X1XXXXXXXXXXXXXX - EVCODE #14 is masked for Program Correlation Messages 1XXXXXXXXXXXXXXX - EVCODE #15 is masked for Program Correlation Messages

42.6.7.5 Development Status Register (DS)

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. The DS register is shown in *Figure 873* and its fields are described in *Table 798*.

Nexus Reg: 0x4 Access: User read only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DBG	LPS			LPC		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

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

Figure 873. Development Status Register (DS)

Table 798. DS Field Descriptions

Field	Description
DBG	CPU Debug Mode Status. 0 CPU not in debug mode. 1 CPU in debug mode.
LPS	LPS Cores System Low Power Mode Status

Table 798. DS Field Descriptions (continued)

Field	Description
LPC	CPU Low-Power Mode Status. 00 Normal (run) mode. 01 CPU in halted state. 10 CPU in stopped state. 11 Reserved.

42.6.7.6 Read/Write Access Control/Status (RWCS)

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 is shown in *Figure 874* and its fields are described in *Table 799*. The RWCS register also provides read/write access status information as shown in *Table 800*.

Nexus Reg: 0x7 Access: User read/write

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

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

Figure 874. Read/Write Access Control/Status Register (RWCS)

Table 799. RWCS Field Description

Field	Description
AC	Access Control. 0 End access. 1 Start access.
RW	Read/Write Select. 0 Read access. 1 Write access.
SZ	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).
MAP	MAP Select. 000 Primary memory map. 001–111 Reserved.

Table 799. RWCS Field Description (continued)

Field	Description
PR	00 Reserved (default to highest access priority) 01 Reserved (default to highest access priority) 10 Reserved (default to highest access priority) 11 Highest access priority
CNT	Access Control Count. Number of accesses of word size SZ.
ERR	Read/Write Access Error. See *Table 800*.
DV	Read/Write Access Data Valid. See *Table 800*.

Table 800 details the status bit encodings.

Table 800. 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

42.6.7.7 Read/Write Access Address (RWA)

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

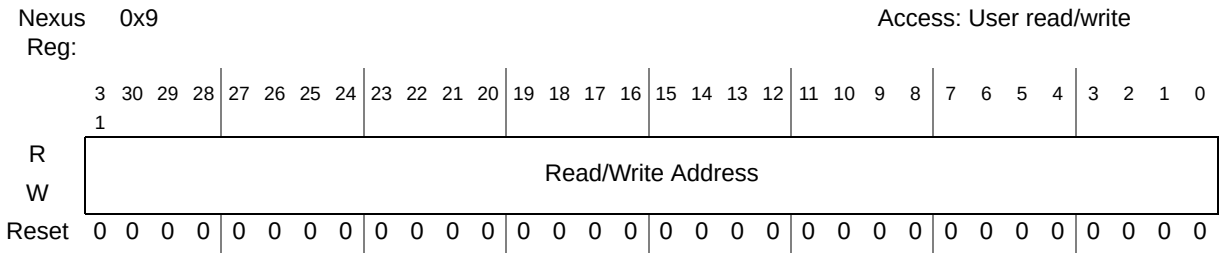


Figure 875. Read/Write Access Address Register (RWA)

42.6.7.8 Read/Write Access Data (RWD)

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

Nexus Reg: 0xA Access: User read/write

	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R W	Read/Write Data							
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

Figure 876. Read/Write Access Data Register (RWD)

Table 801 shows the proper placement of data into the RWD. The “X” in the RWD column indicate byte lanes with valid data.

Table 801. RWD Data Placement for Transfers

Transfer Size and byte offset	RWA[2:0	RWCS[SZ]	RWD			
			31:24	23:16	15:8	7:0
Byte	*xxx*	000	—	—	—	X
Half Word	*xx*0	001	—	—	X	X
Word	*x*00	010	X	X	X	X
Double Word (for burst access only)	000	011				
first RWD pass (low order data)			X	X	X	X
second RWD pass (high order data)			X	X	X	X

Table 802 shows the mapping of RWD bytes to byte lanes of the AHB read and write data buses.

Table 802. RWD data placement for Transfers

Transfer Size and byte offset	RWA[2:0]	RWD			
		31:24	23:16	15:8	7:0
Byte @000	000	—	—	—	AHB[7:0]
Byte @001	001	—	—	—	AHB[15:8]
Byte @010	010	—	—	—	AHB[23:16]
Byte @011	011	—	—	—	AHB[31:24]
Byte @100	100	—	—	—	AHB[39:32]
Byte @101	101	—	—	—	AHB[[47:40]
Byte @110	110	—	—	—	AHB[55:48]
Byte @111	111	—	—	—	AHB[63:56]
Half@000	000	—	—	AHB[15:8]	AHB[7:0]
Half@010	010	—	—	AHB[31:24]	AHB[23:16]
Half@100	100	—	—	AHB[[47:40]	AHB[39:32]
Half@110	110	—	—	AHB[63:56]	AHB[55:48]

Table 802. RWD data placement for Transfers

Transfer Size and byte offset	RWA[2:0]	RWD			
		31:24	23:16	15:8	7:0
Word@000	000	AHB[31:24]	AHB[23:16]	AHB[15:8]	AHB[7:0]
Word@100	100	AHB[63:56]	AHB[55:48]	AHB[[47:40]	AHB[39:32]
Doubleword@000	000				
first RWD pass		AHB[31:24]	AHB[23:16]	AHB[15:8]	AHB[7:0]
second RWD pass		AHB[63:56]	AHB[55:48]	AHB[[47:40]	AHB[39:32]

42.6.7.9 Watchpoint Trigger Register (WT, PTSTC, PTETC, DTSTC, DTETC)

The Watchpoint Trigger Registers allows the watchpoints defined within the Nexus1 logic to trigger actions. These watchpoints can control Program and/or Data Trace enable and disable. The control bits can be used to produce a related “window” for triggering Trace Messages.Watchpoint trigger register WT is used to control triggering by a single selected watchpoint. The Program Trace Start Trigger Control (PTSTC), Program Trace End Trigger Control (PTETC), Data Trace Start Trigger Control (DTSTC), and Data Trace End Trigger Control (DTETC) are used for extended trigger controls for the respective function. If multiple watchpoints are desired for triggering, or a watchpoint beyond watchpoint #13 is required, then one or more of the extended watchpoint trigger registers may be used.

A field encoding of 4’b1111 in one of the WT register fields that enables the corresponding extended trigger register. For all other WT field encodings, the corresponding extended trigger register is disabled and the contents are ignored. When a start trigger is detected, the designated trace features become enabled, and the corresponding enable bits of the DC1 register are set. When a stop trigger is detected, the designated trace features become disabled, and the corresponding enable bits of the DC1 register are cleared. If the same trigger condition is used for both start and stop triggering, then the designated trace features will toggle between being enabled and disabled at each occurrence of the trigger condition. Similarly, if start and stop triggers for a trace feature occur simultaneously, then the designated trace feature will toggle between enabled and disabled depending on the enable state at the time of the trigger events. For example, if tracing is enabled, and a start and stop trigger occur simultaneously, then tracing will be disabled.

The WT register is shown in *Figure 877* and its fields are described in *Table 803*.

Nexus Reg: 0xB — Access: User read/write

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

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

Figure 877. Watchpoint Trigger Register (WT)

Table 803. WT Field Descriptions

Field	Description
PTS	Program Trace Start Control. 0000 Trigger disabled 0001 Use Watchpoint #0 0010 Use Watchpoint #1 .. .. .. 1110 Use Watchpoint #13 1111 Use control settings in the PTSTC register
PTE	Program Trace End Control. 0000 Trigger disabled 0001 Use Watchpoint #0 0010 Use Watchpoint #1 .. .. .. 1110 Use Watchpoint #13 1111 Use control settings in the PTETC register
DTS	Data Trace Start Control. 0000 Trigger disabled 0001 Use Watchpoint #0 0010 Use Watchpoint #1 .. .. .. 1110 Use Watchpoint #13 1111 Use control settings in the DTSTC register
DTE	Data Trace End Control. 0000 Trigger disabled 0001 Use Watchpoint #0 0010 Use Watchpoint #1 .. .. .. 1110 Use Watchpoint #13 1111 Use control settings in the DTETC register

Note: The WT bits can control program/data trace only if the TM bits in the development control register 1 (DC1) have not already been set to enable program and data trace, respectively.

For extended Program Trace start trigger control, the PTSTC register is used.

Nexus Reg: 0x35 Access: User read/write

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

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

Figure 878. Program Trace Start Trigger Control (PTSTC) Register

Table 804. Program Trace Start Trigger Control Register Fields

Field	Description
PTST	PTST – Program Trace Start Trigger Control 00000000000000000000 - Trigger disabled XXXXXXXXXXXXXXXXXXXX1 - Use Watchpoint #0 XXXXXXXXXXXXXXXXXXX1X - Use Watchpoint #1 XXXXXXXXXXXXXXXXXX1XX - Use Watchpoint #2 XXXXXXXXXXXXXXXXX1XXX - Use Watchpoint #3 XXXXXXXXXXXXXXXX1XXXX - Use Watchpoint #4 XXXXXXXXXXXXXXX1XXXXX - Use Watchpoint #5 XXXXXXXXXXXXXX1XXXXXX - Use Watchpoint #6 XXXXXXXXXXXXX1XXXXXXX - Use Watchpoint #7 XXXXXXXXXXXX1XXXXXXXX - Use Watchpoint #8 XXXXXXXXXXX1XXXXXXXXX - Use Watchpoint #9 XXXXXXXXXX1XXXXXXXXXX - Use Watchpoint #10 XXXXXXXXX1XXXXXXXXXXX - Use Watchpoint #11 XXXXXXXX1XXXXXXXXXXXX - Use Watchpoint #12 XXXXXXX1XXXXXXXXXXXXX - Use Watchpoint #13 XXXXXX1XXXXXXXXXXXXXX - Use Watchpoint #14 XXXXX1XXXXXXXXXXXXXXX - Use Watchpoint #15 XXXX1XXXXXXXXXXXXXXXX - Use Watchpoint #16 XXX1XXXXXXXXXXXXXXXXX - Use Watchpoint #17 XX1XXXXXXXXXXXXXXXXXX - Use Watchpoint #18 X1XXXXXXXXXXXXXXXXXXX - Use Watchpoint #19 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #20 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #21

For extended Program Trace end trigger control, the PTETC register is used.

Nexus Reg: 0x36 Access: User read/write

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

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

Figure 879. Program Trace End Trigger Control (PTETC) Register

Table 805. Program Trace End Trigger Control Register Fields

Field	Description
PTET	PTET – Program Trace End Trigger Control 00000000000000000000 - Trigger disabled XXXXXXXXXXXXXXXXXXXX1 - Use Watchpoint #0 XXXXXXXXXXXXXXXXXXX1X - Use Watchpoint #1 XXXXXXXXXXXXXXXXXX1XX - Use Watchpoint #2 XXXXXXXXXXXXXXXXX1XXX - Use Watchpoint #3 XXXXXXXXXXXXXXXX1XXXX - Use Watchpoint #4 XXXXXXXXXXXXXXX1XXXXX - Use Watchpoint #5 XXXXXXXXXXXXXX1XXXXXX - Use Watchpoint #6 XXXXXXXXXXXXX1XXXXXXX - Use Watchpoint #7 XXXXXXXXXXXX1XXXXXXXX - Use Watchpoint #8 XXXXXXXXXXX1XXXXXXXXX - Use Watchpoint #9 XXXXXXXXXX1XXXXXXXXXX - Use Watchpoint #10 XXXXXXXXX1XXXXXXXXXXX - Use Watchpoint #11 XXXXXXXX1XXXXXXXXXXXX - Use Watchpoint #12 XXXXXXX1XXXXXXXXXXXXX - Use Watchpoint #13 XXXXXX1XXXXXXXXXXXXXX - Use Watchpoint #14 XXXXX1XXXXXXXXXXXXXXX - Use Watchpoint #15 XXXX1XXXXXXXXXXXXXXXX - Use Watchpoint #16 XXX1XXXXXXXXXXXXXXXXX - Use Watchpoint #17 XX1XXXXXXXXXXXXXXXXXX - Use Watchpoint #18 X1XXXXXXXXXXXXXXXXXXX - Use Watchpoint #19 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #20 11111111111111111111111 - Use Watchpoint #21

For extended Data Trace start trigger control, the DTSTC register is used.

Nexus Reg: 0x37 Access: User read/write

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

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

Figure 880. Data Trace Start Trigger Control (DTSTC) Register

Table 806. Data Trace Start Trigger Control Register Fields

Field	Description
DTST	DTST - Data Trace Start Trigger Control 00000000000000000000 - Trigger disabled XXXXXXXXXXXXXXXXXXXX1 - Use Watchpoint #0 XXXXXXXXXXXXXXXXXXX1X - Use Watchpoint #1 XXXXXXXXXXXXXXXXXX1XX - Use Watchpoint #2 XXXXXXXXXXXXXXXXX1XXX - Use Watchpoint #3 XXXXXXXXXXXXXXXX1XXXX - Use Watchpoint #4 XXXXXXXXXXXXXXX1XXXXX - Use Watchpoint #5 XXXXXXXXXXXXXX1XXXXXX - Use Watchpoint #6 XXXXXXXXXXXXX1XXXXXXX - Use Watchpoint #7 XXXXXXXXXXXX1XXXXXXXX - Use Watchpoint #8 XXXXXXXXXXX1XXXXXXXXX - Use Watchpoint #9 XXXXXXXXXX1XXXXXXXXXX - Use Watchpoint #10 XXXXXXXXX1XXXXXXXXXXX - Use Watchpoint #11 XXXXXXXX1XXXXXXXXXXXX - Use Watchpoint #12 XXXXXXX1XXXXXXXXXXXXX - Use Watchpoint #13 XXXXXX1XXXXXXXXXXXXXX - Use Watchpoint #14 XXXXX1XXXXXXXXXXXXXXX - Use Watchpoint #15 XXXX1XXXXXXXXXXXXXXXX - Use Watchpoint #16 XXX1XXXXXXXXXXXXXXXXX - Use Watchpoint #17 XX1XXXXXXXXXXXXXXXXXX - Use Watchpoint #18 X1XXXXXXXXXXXXXXXXXXX - Use Watchpoint #19 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #20 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #21

For extended Data Trace end trigger control, the DTETC register is used.

Nexus Reg: 0x38 Access: User read/write

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

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

Figure 881. Data Trace End Trigger Control (DTETC) Register

Table 807. Data Trace End Trigger Control Register Fields

Field	Description
DTET	DTET – Data Trace End Trigger Control 00000000000000000000 - Trigger disabled XXXXXXXXXXXXXXXXXXXX1 - Use Watchpoint #0 XXXXXXXXXXXXXXXXXXX1X - Use Watchpoint #1 XXXXXXXXXXXXXXXXXX1XX - Use Watchpoint #2 XXXXXXXXXXXXXXXXX1XXX - Use Watchpoint #3 XXXXXXXXXXXXXXXX1XXXX - Use Watchpoint #4 XXXXXXXXXXXXXXX1XXXXX - Use Watchpoint #5 XXXXXXXXXXXXXX1XXXXXX - Use Watchpoint #6 XXXXXXXXXXXXX1XXXXXXX - Use Watchpoint #7 XXXXXXXXXXXX1XXXXXXXX - Use Watchpoint #8 XXXXXXXXXXX1XXXXXXXXX - Use Watchpoint #9 XXXXXXXXXX1XXXXXXXXXX - Use Watchpoint #10 XXXXXXXXX1XXXXXXXXXXX - Use Watchpoint #11 XXXXXXXX1XXXXXXXXXXXX - Use Watchpoint #12 XXXXXXX1XXXXXXXXXXXXX - Use Watchpoint #13 XXXXXX1XXXXXXXXXXXXXX - Use Watchpoint #14 XXXXX1XXXXXXXXXXXXXXX - Use Watchpoint #15 XXXX1XXXXXXXXXXXXXXXX - Use Watchpoint #16 XXX1XXXXXXXXXXXXXXXXX - Use Watchpoint #17 XX1XXXXXXXXXXXXXXXXXX - Use Watchpoint #18 X1XXXXXXXXXXXXXXXXXXX - Use Watchpoint #19 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #20 1XXXXXXXXXXXXXXXXXXXX - Use Watchpoint #21

42.6.7.10 Nexus Watchpoint Mask Register (WMSK)

The Nexus Watchpoint Mask register controls which watchpoint events are enabled to produce Watchpoint Trace Messages (DC1[TM] must also be programmed to generate Watchpoint Trace Messages).

For extended Data Trace end trigger control, the DTETC register is used.

Nexus Reg: 0x33 Access: User read/write

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

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

Figure 882. Watchpoint Mask Register

Table 808. Watchpoint Mask Register Fields

Field	Description
WEM	WEM - Watchpoint Enable for Messaging 00000000000000000000 - No Watchpoints enabled for Watchpoint Trace Messaging XXXXXXXXXXXXXXXXXXXX1 - Watchpoint #0 enabled for WTM XXXXXXXXXXXXXXXXXXX1X - Watchpoint #1 enabled for WTM XXXXXXXXXXXXXXXXXX1XX - Watchpoint #2 enabled for WTM XXXXXXXXXXXXXXXXX1XXX - Watchpoint #3 enabled for WTM XXXXXXXXXXXXXXXX1XXXX - Watchpoint #4 enabled for WTM XXXXXXXXXXXXXXX1XXXXX - Watchpoint #5 enabled for WTM XXXXXXXXXXXXXX1XXXXXX - Watchpoint #6 enabled for WTM XXXXXXXXXXXXX1XXXXXXX - Watchpoint #7 enabled for WTM XXXXXXXXXXXX1XXXXXXXX - Watchpoint #8 enabled for WTM XXXXXXXXXXX1XXXXXXXXX - Watchpoint #9 enabled for WTM XXXXXXXXXX1XXXXXXXXXX - Watchpoint #10 enabled for WTM XXXXXXXXX1XXXXXXXXXXX - Watchpoint #11 enabled for WTM XXXXXXXX1XXXXXXXXXXXX - Watchpoint #12 enabled for WTM XXXXXXX1XXXXXXXXXXXXX - Watchpoint #13 enabled for WTM XXXXXX1XXXXXXXXXXXXXX - Watchpoint #14 enabled for WTM XXXXX1XXXXXXXXXXXXXXX - Watchpoint #15 enabled for WTM XXXX1XXXXXXXXXXXXXXXX - Watchpoint #16 enabled for WTM XXX1XXXXXXXXXXXXXXXXX - Watchpoint #17 enabled for WTM XX1XXXXXXXXXXXXXXXXXX - Watchpoint #18 enabled for WTM X1XXXXXXXXXXXXXXXXXXX - Watchpoint #19 enabled for WTM 1XXXXXXXXXXXXXXXXXXXX - Watchpoint #20 enabled for WTM 1XXXXXXXXXXXXXXXXXXXX - Watchpoint #21 enabled for WTM

42.6.7.11 Overrun Control Register (OVCR)

Nexus Overrun Control register controls the Nexus behavior as the internal message queues fill up. Response options include suppressing selected message types, or stalling processor instruction execution. The OVCR register is shown in *Figure 883* and its fields are described in *Table 809*.

Nexus Reg: 0x4 Access: User read only

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

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

Figure 883. Overrun Control Register (OVCR)

Table 809. Overrun Control Register Fields

Field	Description
SPTHOLD	Suppression Threshold 00 Suppression threshold is when message queues are 1/4 full 01 Suppression threshold is when message queues are 1/2 full 10 Suppression threshold is when message queues are 3/4 full 11 Reserved
SPEN	Suppression Enable 000000 Suppression is disabled xxxxx1 Ownership Trace message suppression is enabled xxxx1x Data Trace message suppression is enabled xxx1xx Program Trace message suppression is enabled xx1xxx Watchpoint Trace message suppression is enabled x1xxxx Reserved 1xxxxx Data Acquisition message suppression is enabled
STTHOLD	Stall Threshold 00 Stall threshold is when message queues are 1/4 full 01 Stall threshold is when message queues are 1/2 full 10 Stall threshold is when message queues are 3/4 full 11 Reserved
STEN	Stall Enable 0 Stalling is disabled 1 Stalling is enabled

42.6.7.12 Data Trace Control Register (DTC)

The data trace control register controls whether DTM messages are restricted to reads, writes, or both for a user programmable address range. Two data trace channels are controlled by the DTC for the Nexus3 module. Each channel can also be programmed to trace data accesses or instruction accesses.

Nexus Reg: 0xD Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RWT1		RWT2		RWT3		RWT4		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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RC1	RC2	RC3	RC4	DI	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 884. Data Trace Control Register (DTC)

Table 810 details the data trace control register fields.

Table 810. DTC Field Description

Field	Description
31–30 RWT1	Read/write trace 1. 00 No trace enabled. *x*1 Enable data read trace. 1*x* Enable data write trace.
29–28 RWT2	Read/write trace 2. 00 No trace enabled. *x*1 Enable data read trace. 1*x* Enable data write trace.
27–26 RWT3	Read/write trace 3. 00 No trace enabled. *x*1 Enable data read trace. 1*x* Enable data write trace.
25–24 RWT4	Read/write trace 4. 00 No trace enabled. *x*1 Enable data read trace. 1*x* Enable data write trace.
7 RC1	Range control 1. 0 Condition trace on address within range. 1 Condition trace on address outside of range.
6 RC2	Range control 2. 0 Condition trace on address within range. 1 Condition trace on address outside of range.
5 RC3	Range control 2. 0 Condition trace on address within range. 1 Condition trace on address outside of range.
4 RC4	Range control 2. 0 Condition trace on address within range. 1 Condition trace on address outside of range.
3 DI	Data access/instruction access trace 1. 0 Condition trace on data accesses. 1 Condition trace on instruction accesses.

42.6.7.13 Data Trace Start Address Registers (DTSA1—DTSA2)

The data trace start address registers define the start addresses for each trace channel.

Nexus Reg: (0xE DTSA1, 0XF DTSA2, 0x10 DTSA3, 0x11 DTSA4) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace Start Address																															
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

Figure 885. Data Trace Start Address Register 1– 4 (DTSA1–DTSA4)

42.6.7.14 Data Trace End Address Registers (DTEA1—DTEA2)

The data trace end address registers define the end addresses for each trace channel.

Nexus Reg: (0x12 DTEA1, 0x13 DTEA2, 0x14 DTEA3, 0x15 DTEA1) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace End Address																															
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

Figure 886. Data Trace End Address Register 1 – 4 (DTEA1–DTEA4)

Table 811 illustrates the range that selected for data trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 811. Data Trace—Address Range Options

Programmed Values	Range Control Bit Value	Range Selected
DTSA < DTEA	0	DTSA → ←DTEA
DTSA < DTEA	1	← DTSA DTEA →
DTSA > DTEA	N/A	Invalid range—no trace
DTSA = DTEA	N/A	Invalid range—no trace

Note: DTSA must be less than DTEA in order to guarantee correct data write/read traces. Data trace ranges are inclusive of the DTSA and DTEA addresses for Range Control settings indicating "within range", and are exclusive of the DTSA and DTEA addresses for Range Control settings indicating "outside of range."

42.6.8 Register Access via JTAG / OnCE

Access to Nexus3 register resources is enabled by loading a single instruction (ACCESS_AUX_TAP_Z0) into the JTAG Instruction Register (IR) (OnCE OCMD register), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS

instruction (refer to *Table 776*). Access to Nexus3+ register resources is enabled by loading a single instruction (ACCESS_AUX_TAP_Z4) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (refer to *Table 776*). For the Nexus3 module, the OCMD value is 0b00_0111_1100.

Reading/writing of a Nexus register requires two passes through the data-scan (DR) path of the JTAG state machine

1. The first pass through the DR selects the Nexus register to be accessed by providing an index, and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:

(7 bits)	(1 bit)
Nexus Register Index	R/W

RESET Value: 0x00

Nexus Register Index:	Selected from values in *Table 793*
Read/Write (R/W)	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, LSB first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the CAPTURE-DR state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the UPDATE-DR state.
      – Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
   – Data Trace, Processor overrun control

42.7 Debug Implementation

This section describes the practical implementation of the debug port, its management, and the multiplexing strategy of its pads. It is required on SPC564Bxx and SPC56ECxx to support the Class 3+(2010) debug features on the e200Z4d and minimum Class3(2010) on the e200Z0h.

42.8 Debug Capabilities

SPC564Bxx and SPC56ECxx supports the Class 3 and Class 3+ debug features listed in *Table 812*

Table 812. Nexus3 Debug Requirements

Feature	Class 1	Class 2	Class 3	Class 4
Static debug	X	X	X	X
Set breakpoints/watchpoints	X	X	X	X
Watchpoint messaging	X	X	X	X
Ownership trace	—	X	X	X
Program trace (BTM)	—	X	X	X
Port replacement	—	optional	optional	optional
Data trace (write only)	—	—	X	X
Dynamic memory read/write	—	—	X	X
Data trace (read/write)	—	—	optional	optional
Data acquisition	—	—	optional	optional
Overrun Control	—	—	—	X
Memory substitution	—	—	—	X
Complex triggering	—	—	—	X
External memory substitution	—	—	—	optional

These debug capabilities are available at room temperature (25° C) and high temperature (125° C) when the SPC564Bxx and SPC56ECxx is at maximum speed.

The JTAG and Nexus port will work both at 3.3V and 5V , according to the device power supply.

Based on the following assumptions:

- Average message length: 20 bits
- 1 jump every 10 instructions
- MCKO capable of supporting 1/2 the system clock frequency (64 MHz maximum)
- MDO pins operate at half of the system clock speed

It is required to have 12 dedicated medium MDO pins. The N3/N3+ auxiliary port is bonded out only in the the BGA 256 package.

Boundary scan test is supported.

42.9 Debug Port

The debug port is composed of a total number of 25 pads. For Nexus3+ pin description, see *Chapter 4: Signal Description*.

The reset and ready pins that are often present as extensions to the JTAG port are not implemented.

42.9.1 Nexus3+ Auxiliary Port

The N3(+) port provides real-time development class 3+ capabilities in compliance with the IEEE-ISTO 5001-2010 standard. This development support is supplied without requiring external address and data pins for internal visibility.

By default, after power-on reset, N3(+) circuitry (controller) and the dedicated pad are disabled to avoid power consumption. It can only be enabled via a certain sequence given by the debugger. As soon as the N3(+) port is enabled, the Nexus pads are enabled.

Note: Nexus pads are configured on the lowest power mode on non-emulation packages.

- *A full port (12MDOs) and reduced port mode (8MDOs) shall be supported and controlled via the Port Configuration Register (PCR) register.*

43 Revision history

Table 813. Document revision history

Date	Revision	Changes
17-Dec-2010	1	Initial release
02-Aug-2011	2	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 SPC564Bxx and SPC56ECxx microcontroller family”. Renamed “Feature summary” to “Feature details”. Moved the “Memory map” section to its own separate chapter. Deleted the duplicate device-comparison tables. In the device-comparison table, added the text “There is a configurable e200z0 system clock divider for this purpose.” to footnote 3. In the block diagram, changed PIT to PIT_RTI. In the “Feature summary” section: – Changed “TLB” to “translation lookaside buffer (TLB)”. – Revised the “System clocks and clock generation” section. – Revised the SIUL section. – Revised the “On-chip SRAM” section. – Changed PIT to PIT_RTI. – Revised the DSPI section. – In the LINFlexD section, changed “Up to 10” to “10”. – In the LINFlexD section, changed “Configurable Break duration of up to 36 bit times” to “Configurable break duration of up to 50 bit times”. – In the FlexCAN section, changed “Up to 6” to “6”. – Revised the PIT_RTI section. – In the STM section, deleted “Instantiated in the same CPU clock domain”. – Revised the RTC/API section. – In the NDI section, changed “bandwidth will be limited” to “bandwidth will be limited in RPM”. – In the “System clocks and clock generation” section, changed “Programmable output clock divider of system clock” to “Programmable divider for output clock”.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Introduction (continued) – In the XBAR section, deleted “32-bit internal address bus for e200z0h, 64-bit internal data bus for e200z4d”. – Revised the “Flash memory” section. Added the “How to use the SPC564Bxx and SPC56ECxx documents” and “Using the SPC564Bxx and SPC56ECxx” sections. Chapter Memory Map Added this chapter (content previously contained in the Overview chapter). Changed “Test Sector Data Flash Array 0” to “Data flash memory array 0 test sector” Chapter Signal Description Revised the pinout information for the 176-pin QFP. Changed WKUP to WKPU to match the block abbreviation. Changed ANS to ADC0_S or ADC1_S (as appropriate). Revised the footnotes in the “Functional port pin descriptions” table. In the “System pin descriptions” table, added a footnote to the A pads regarding not using IBE. For ports PB[12–15], changed ANX to ADC0_X. Revised the presentation of the ADC functions on the following ports: – PB[4–7] – PD[0–11] In the “System pin descriptions” table, swapped the function description for EXTAL and XTAL. In the “Functional port pin descriptions” table, changed “ALT” to “AF”. For port PA[0], added CAN1RX. In the “Functional port pin descriptions” table, added a footnote about multiple inputs to the “I/O direction” column. Chapter Safety Migrated the chapter contents to the “Register Protection” chapter. Chapter Microcontroller Boot Added this chapter. Chapter Clock Description 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 Added the following text above the peripheral clock divider setup table: “Dynamic switching of the clock dividers for the peripherals takes effect immediately and affects the external functions.”.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Clock Description (continued) In the FIRC “Functional description” section, changed “provided by RC_CTL[FIRC_STDBY] bit” to “provided by RC_CTL[FIRCON_STDBY] bit”. In the SXOSC_CTL[OSCON] field description, changed “powerdown control” to “enable”. 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. 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 FMPLL CR[S_LOCK] field description, changed the note (was “S_LOCK =1 signals coarse lock. The system clock should not be changed to PLL output for at least 200 ms after S_LOCK is set.”, is “SLOCK=1 indicates that the FMPLL has achieved coarse lock. Fine lock is achieved 200 µs after the FMPLL is enabled.”). In the CMU Introduction section, changed “clock management unit” to MC_CGM. In the FMPLL section, deleted the duplicate “FMPLL memory map” table. In the “Clock gating” section, added a note about altering the e200z0 clock divider. In the “Crystal clock monitor” section, added a note about the function of the XOSC monitor. In the “FMPLL clock monitor” section, added a note about the function of the FMPLL monitor. Clock Generation Module Revised the reset values of the CGM_SC_DC*n* registers to show that the DE*n* fields reset to 1and the DIV*n* fields reset to 0. In the CGM_AC1_SC section, deleted “undivided: (unused)”. Replaced “Z0” with “e200z0h”. Revised the note in the CGM_Z0_DCR section. Revised the CGM_FEC_DCR section. Revised the CGM_FLASH_DCR section. Revised the CGM_AC0_SC[SELCTL] field description. Revised the “MC_CGM Auxiliary Clock 0 Generation Overview” figure. In the “MC_CGM Auxiliary Clock 1 Generation Overview” figure, deleted TCK. Changed the CGM_OCDS_SC[SELCTL] reset value (was 0b0001, is 0b0000). Changed the CGM_AC1_DC0 reset value (was 0b0000_0001, is 0b1000_0000). In the CGM_AC1_SC[SELCTL] field description, added a note about disabling the FlexRay module.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Mode Entry Module Changed WARNING to CAUTION. Changed the ME_HALT0_MC[CFLAON] reset value (was 0b10, is 0b11). Revised the ME_GS field descriptions. In the ME_STOP0_MC register figure, changed the FMPLLON, FXOSCON, FIRCON, and SYSCLK fields to be read/write (were read-only). In the ME_PS1 register, changed field name S_I2C_DMA to S_I2C. Revised the ME_PSn[S_<periph>] field description. Revised the Overview section. Revised the Features section. Revised the “Modes of operation” section. Changed HALT0 to HALT. Changed STOP0 to STOP. Changed STANDBY0 to STANDBY. In the “Register description” section, deleted “The bytes are ordered according to big endian”. Revised the text description of the ME_MCTL register. Revised the ME_ME section. In the ME_IS[I_IMODE] field description, added information about the events that can cause invalid mode interrupts. Changed “FIRC (16 MHz internal RC osccillator)” to “FIRC”. In the ME_RESET_MC section, changed “configures system behavior during RESET mode” to “details the mode configuration during reset”. Extensive revisions throughout the “Functional description” section and subsections. In the ME_GS[S_FMPLL] field description, added a note about coarse and fine lock. Changed the ME_HALT_MC[MVRON] field to read-only (was read/write). In the “Mode Configuration Registers (ME_<mode>_MC) field descriptions” table, added a note to the MVRON field about clearing it in STOP or HALT modes. In the “Mode Configuration Registers (ME_<mode>_MC) field descriptions” table, added a note to the DFLAON field describing the dependence on CFLAON. In the ME_<mode>_MC[DFLAON] field description, added a note about configuring reset sources as long resets. Added the “Peripheral control registers by peripheral” table. In the “STANDBY Mode” section, added a note about enabling the WKPU clock. Chapter Reset Generation Module In the RGM_STDBY[BOOT_FROM_BKP_RAM] field description, added “(when using the e200z4d core from RAM in STANDBY0 ensure that VLE code is used, as described in this section)” to the description of value 1. Revised the RGM_DERD section to indicate that the register is always read-only. Changed WARNING to CAUTION. Changed “Z0” to “e200z0h”. Changed “Z4” to “e200z4d”.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Revised the “Reset Sources” section. Renamed the RGM_STDBY register (was “STANDBY0 Reset Sequence”, is “STANDBY Reset Sequence”. Revised the RGM_FEAR[AR_CMU_OLR] field description. In the “IDLE Phase” section, changed “control of the system to the platform” to “control of the chip to the core”. Chapter Reset Generation Module (continued) Revised the “Boot Mode Capturing” section. Revised the RGM_FES[F_CMU_FHL] field description. Changed STANDBY0 to STANDBY. Revised the RGM_FES[F_CORE] field description. Changed “core reset” to “debug control core reset”. Chapter Power Control Unit Revised the PCU_PCONF2..3 section. Added the “DRUN, SAFE, TEST, RUN0..3, HALT0, and STOP0 Mode Transition” section. In the “STANDBY0 Mode Transition” section, changed “STANDBY0 offers...” to “STANDBY offers...”. Revised the Overview section. Changed HALT0 to HALT. Changed STOP0 to STOP. Changed STANDBY0 to STANDBY. Revised the “RAM configurations in modes” table. In the “STANDBY Mode Transition” section, added “Prior to standby entry, PCTL for WKPU should be disabled”. Chapter Voltage Regulators and Power Supplies Changed STANDBY0 to STANDBY. Revised the last sentence in the “High power regulator (HPREG)” section. Revised the “Memory map” section. Chapter Wakeup Unit Changed the last source for interrupt vector 3 (was PJ11, is PJ13). In the Overview section: – Deleted EIRQn. – Replaced the list of wakeup lines with a table. – Added the note “In HALT mode and in STOP mode where the system clock is still enabled, an external interrupt (EIRQ) or any peripheral interrupt can be used to wake the device up”. Revised the reset value of NCR. Moved the note in the “External signal description” section to the Overview section, and deleted the “External signal description” section. In the note 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. Revised the NCR section.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Real Time Clock / Autonomous Periodic Interrupt Changed the reset state of bit 0 of RTC Status Register from a 1 to a 0. Change the reset state of bit 0 of RTC Count Register for a 1 to a 0.. Added notes in FEATURES list to clarify clock sources and their divider chains. Corrected bit ordering issues in .RTC/API Clock diagram. Delected section Device-specific information. Deleted referenced to _input isolation" in section on Modes of Operation. Updated operation of RTC counter in Debug Mode. Changed the reset state of the CNT EN bit in the RTCC register to a logic 0. Updated operation of FRZEN bit and APIVAL bit in RTCC register field descriptions. Changed all Reserved bits in RTC Status Register and RTC Counter Register to a logic 0. Updated descriptions in RTC functional description for clarity. In the "Functional mode" section, changed "bus interface is disabled" to "bus interface is disabled and no configuration changes are permitted". 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 memory 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. In the "Internal multiplexer correspondence" table, revised the entries in the "Rx selected" column. Chapter Enhanced Direct Memory Access In the "TCDn field descriptions", revised the INT_HALF description. In the EDMA_CR section, added the GRP1PRI field. In the "eDMA memory map" table, added the module base address. Chapter eDMA Channel Multiplexer Changed PIT to PIT_RTI. Changed the chapter title (was "DMA Channel Multiplexer", is "eDMA Channel Multiplexer") and changed "DMA" to "eDMA" as appropriate to match the title. Revised the chapter to show that periodic triggering mode is available for channel-0 to channel-3 (not for channel 1-4). In the CHCONFIG register figure, revised the bit order (was 7..0, is 0..7) to match Power Architecture convention. In the "DMA channel mapping" table, changed the entries for DMA_MUX channels 19–22 (were EMIOS1..., are EMIOS0...). In the "DMA_MUX memory map" table, added the module base address. Chapter Interrupt Controller Changed the chapter title (was "Interrupts and Interrupt Controller", is "Interrupt Controller"). Revised the INTC_IACKR_PRCn sections to illustrate the registers' dependence on INTC_MCR[VTES] more clearly.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	In the INTC_EOIR_PRC1 figure, changed the offset (was 0x0018, is 0x001C). Changed WKUP to WKPU. Changed PIT to PIT_RTI. In the “INTC memory map” table, added the module base address Chapter Crossbar Switch Changed the chapter title (was “Multi-Layer AHB Crossbar Switch”, is “Crossbar Switch). In the “Master/slave mappings” table, changed “Nexus3” to “Nexus3+”. Deleted content for Alternate Master Priority Registers. Deleted content for Alternate Slave General Purpose Control Registers. In the “XBAR block diagram” figure, changed “Cold” to “Code”. Added content to the “Priority elevation” section. In the Overview section, changed “up to eight simultaneous connections” to “up to 5 simultaneous connections”. In the “XBAR block diagram” figure, added master and slave numbers. In the “Limitations” section, deleted the paragraph about port compliance. In the MPR figure, added reset values. Deleted the “XBAR Master Port Block Diagram” figure. Deleted the “XBAR Slave Port Block Diagram” figure. Revised the “Features” section. In the Overview section, deleted “generic multi-layer AHB”. Revised the “General operation” section. Revised the “Register summary” section. Revised the MPRn section. Revised the SGPCRn section. Revised the MGPCRn section. Revised the “Coherency” section. Revised the “Fixed priority operation” section. In the “Priority assignment” section, changed “(MPR or AMPR)” to “(MPR)”. Deleted the “Context switching” section. In the “XBAR memory map” table, added the module base address. Chapter Memory Protection Unit In the MPU_CESR[SPERR] field description, changed: – SPERR[0] to SPERR[7] – SPERR[1] to SPERR[6] – SPERR[2] to SPERR[5] – SPERR[3] to SPERR[4] – SPERR4] to SPERR[3] In the Features section, changed the entry for access control definitions. In the “Memory map and register description” section, changed “up to 3 AHB slave ports” to “up to 5 AHB slave ports”. In the “MPU memory map” table, added the module base address. Chapter Semaphores Changed “Z0” to “e200z0h”. Changed “Z4” to “e200z4d”. Deleted the note in the Features section. In the “Semaphores memory map” table, added the module base address.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Performance Optimization Added this chapter. Chapter Performance Optimization Added this chapter. Chapter System Integration Unit Lite Added the “PISR / ISPx muxing configuration” table. Added the steps to select IPSx Mux. Changed “WARNING” to “CAUTION”. Revised the PCRx[WPE] and PCRx[WPS] field descriptions. Updated SUIL memory map to reflect byte addressing. Corrected IRE in IRER Register to EIRE. In the “PCR bit implementation by pad type” table, created a standalone entry for pad type S (with the SRC field reserved). Deleted the MIDR1[CSP] field (is reserved). In the “Peripheral input pin selection” table, added an entry for PSMI0_3 > PADSEL0 > 0x500 (101, PCR[0]). Rewrote the entire PISR section. Revised the “MIDR2 field descriptions” table to show how to calculate total flash memory size. In the PSMI section, revised the introductory text. In the “Peripheral input pin selection” table, revised the entry for PSMI48_51 > PADSEL50 > 001 (was PCR[48], is PCR[148]). In the “SIUL memory map” table, added the module base address. Chapter Inter-Integrated Circuit Bus Controller Module Added a note below Memory Map that I2C registers are accessible in the Supervisor mode. Added DMA Application Information. 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 IBSR figure, changed the IBAL and IBIF fields to w1c. In the IBIC[BIIE] field description, added a note about when this bit can be set. 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 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter LIN Controller (LINFlexD) In the register figures: – Added “Access: User read/write” to all register figures. – Updated instances of “These fields are writable only in Initialization mode.” to “These fields are writable only in Initialization mode (LINCR1[INIT] = 1).”. In the LINESR figure, changed the footnote “If LINTCSR[LTOM] is set, these fields are read-only.“ to read “If LINTCSR[LTOM] = 1, these fields are read-only.“ In the LINTOCR figure, added the footnote “The HTO field can only be written in slave mode, LINCR1[MME] = 1“. Added missing content to the IFMI and IFMR register figures. In the “Filter submodes” section, changed “eight IFCR registers” to “16 IFCRs” and “eight identifiers” to “16 identifiers”. In the “9-bit data frame” section, changed “The 8-bit UART data frame” to “The 9-bit UART data frame” and “sum of the 7 data bits” to “sum of the 8 data bits”. Added missing content to the LINTCSR and BIDR register figures. Revised the IFER section. Added content to the “8-bit timeout counter” section. Added the “Error calculation for programmed baud rates” table. In the LINCR1[MME] field description, changed “Master and Slave mode enable” to “Slave mode enable”. In the “LIN mode features” section, changed “with as clock source” to “with FIRC as clock source”. Revised the “Memory map and register description” section to show the differences in register availability on the various LINFlexD modules on this chip. In the LINCR1[BF] field description, changed “this bit is reserved” to “this bit is reserved and always reads 1”. In the DMATXE register, changed bits 16–30 to reserved. In the GCR[SR] field description, added “This field should be cleared by software to perform further operations (the field is not cleared by hardware)”. Changed “kbps” to “Kbit/s”. In the “TCD chain memory map (master node, TX mode)” figure, changed the second instance of “Extended Frame (n+2)” to “Extended Frame (n+3)”. In the “TCD chain memory map (master node, RX mode)” figure, changed the second instance of “Extended Frame (n+1)” to “Extended Frame (n+2)”. In the “TCD chain memory map (slave node, TX mode)” figure, changed the second instance of “Extended Frame (n+1)” to “Extended Frame (n+2)”. In the “TCD chain memory map (slave node, RX mode)” figure, changed the second instance of “Extended Frame (n+1)” to “Extended Frame (n+2)”. Chapter FlexCAN Changed the chapter title (was “FlexCAN module”, is “FlexCAN”). Added a note in the description of CLK_SRC bit of the Control Register (CTRL) description. Deleted references to Stop mode (not supported on this chip). In the CTRL field descriptions, added “0” and “1” to indicate what the bit values of 0 and 1 mean, respectively. In the “Modes of operation” section, revised the description of Module Disable mode. Revised the “Module Disable mode” section. In the “FlexCAN memory map” table: – Revised the addresses for RXIMR0–63. – Added the module base addresses.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Deserial Serial Peripheral Interface In the “DSPI configuration” table, changed “No of CS supported” for DSPI 5 (was 2, is 3). Deleted references to DSPI_HCR. In the DSPI_CTAR section, deleted “The number of CTAR registers is parameterized in the RTL and can be from two to six registers.”. In the DSPI_CTARn[LSBFE] field description, deleted “When operating in TSB configuration, this bit should be set to be compliant to MSC specification.”. In the “Continuous selection format” section, added a note about filling the TX FIFO. In the “Continuous serial communications clock” section, revised the rules. In the DSPI_TCR, renamed the register field (was SPI_TCNT, is TCNT). In the DSPI_RSER, removed underscores from field names. In the DSPI_DSICR, changed FMSZ[4] to FMSZ4. Deleted references to “SoC specific” content. In the memory map table, added the module base addresses. Chapter FlexRay Communication Controller In the “Channel assignment description” table, changed the entries for CHA=1 / CHB=1 / dynamic segment (are “reserved; functionality not guaranteed”). In the FR_MCR[CLKSEL] field description, added a note about disabling the FlexRay module before changing clock sources. In the “FlexRay memory map” table, added the module base address. Chapter Fast Ethernet Controller Added the ECR figure. In the “FEC register map” table, added the module base address. Chapter Timers Added this chapter (incorporates content from STM, eMIOS, and PIT_RTI chapters). Chapter Analog-to-Digital Converter Changed PIT to PIT_RTI. Added separate Channel Watchdog Select Registers for all ADC0 and ADC1 channels. Added the “ADC channel mapping” table. In the “Device-specific implementation” section, added the “ADC_0 mux control signal availability” table. In the following sections, changed “32 to 60 (standard channels)” to “32 to 63 (standard channels)” and added ADC_0/ADC_1 designations: – CEOCFR – CIMR – DMAR – Threshold Register – PSR – CTR – NCMR – JCMR

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Analog-to-Digital Converter (continued) – Data registers – CWSELR – AWORR Revised the CDR figure for ADC_1. Added meaningful descriptions to the CTR[INPCMP] field description. In the NCMR section, added a note about internal channel priority. Revised the “ADC_0 mux control signal availability” table. In the Introduction section, added a note about configuring GPIOs. 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). Changed PIT to PIT_RTI. In the “CTU memory map” table, added the module base address. Chapter Flash Memory Updated CF and DF feature list to clarify RWW operation Shadow sector structure table: Removed NVSRC. Corrected address offsets of many CF registers. Corrected Flash multi module sectorization table. Deleted the duplicate register maps. Deleted references to “optimized” memory. Revised the NVUSRO definition to show that the WATCHDOG_EN bit is bit 0 (not 31), the PAD_3V5V bits are bits 2–3 (not 29–28), and the STCU_EN bit is bit 21 (not 10). Revised the code flash memory MCR field descriptions. Revised the code flash memory LML field descriptions. Revised the Platform Flash Controller Introduction, Overview, and Features sections. Revised the Platform Flash Controller “External Signal Descriptions” section. In the “Memory map and register description” section, in the code examples, changed “_bfen” to “_bfe”. Revised the PFCR0 section. Revised the PFCR1 section. Changed “Bank 0 and 2 Page Read Buffers and Prefetch Operation” to “Code flash memory bank 0 and 2 page read buffers and prefetch operation”. Changed “Buffer Allocation” to “Code flash memory buffer allocation”. In the NVPWD1 section, added a note about reading the RCHW. In the NVUSRO[PAD3V5V[0]] field description, changed $V_{DD_HV_A}$ to $V_{DD_HV_B}$. In the NVUSRO[PAD3V5V[1]] field description, changed $V_{DD_HV_B}$ to $V_{DD_HV_A}$. In the PFCR1[B1_RWSC] field description, deleted “The integrator ... results”. In the code flash memory MCR section, added content about Stall/Abort-While-Write. Deleted references to “SoC specification”. In the PFCR0[B02_RWWC] field description, changed the entry for 0-- (is “This state should be avoided”). In the PFCR1[B1_RWWC] field description, added an entry for 0--. Changed “Nonvolatile System Censorship Information (NVSCI)” to “Nonvolatile

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Flash Memory (continued) System Censorship Control (NVSCC)". In the NVUSRO figure, changed bit 2 from PAD_3V5V[1] to PAD_3V5V[0] and bit 3 from PAD_3V5V[0] to PAD_3V5V[1]. In the PFCR0[B02_RWWC] field description, added "Setting to this state can cause unpredictable operation" to the description for 0--. In the PFCR1 section, added a note about not updating these registers directly. Chapter Static RAM In the Introduction section, changed "retain 8 KB, 64 KB, or 96 KB" to "retain 8 KB, 40 KB, 64 KB, or 96 KB". Chapter Register Protection Added this chapter. Chapter Software Watchdog Timer In the Features section, added "The SWT is clocked by the SIRC". In the "SWT memory map" table, added the module base address. Chapter Error Correction Status Module Removed XBAR_ARB bit from the MUDCR register. 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. Renamed IOPMC to IMC. Revised the Introduction section. Revised the Features section. Revised the "ECC registers" section. In the "ECSM memory map" table, fixed the reserved entry at 0x0C (ends at 0x1E, not 0x1F). Revised the EEGR[ERRBIT] field description. In the "ECSM memory map" table, added the module base address. Chapter Self-Test Control Unit In the "STCU main features" section, added "The STCU cannot be started by software." In the "STCU register map" table, added the module base address. Deleted the STCU_RUN register and the associated reference in the "Self-Test sequence after reset trigger" section. Chapter Cryptographic Services Engine In the "CSE memory map" table, added the module base address. Chapter JTAG Controller Removed references to JCOMP. Revised the "JTAG STL (IEEE 1149.1) block diagram" figure.

Table 813. Document revision history

Date	Revision	Changes
02-Aug-2011	2 (continued)	Chapter Nexus Development Interface Added the “NPC_HNDSHK module” section. Changed “Z4d” to “e200z4d” and “Z0h” to “e200z0h” in NDI Functional Block Diagram. In the PCR[NEXCFG] field description, deleted “Function is SoC specific”. Changed STANDBY0 to STANDBY. Changed HALT0 to HALT. Changed STOP0 to STOP. Deleted references to the emulation package. In the DID[PIN] field description, changed 0x248 to 0x249. In the “PCR field descriptions” table, added a footnote to LP_DBG_EN and LPn_SYN about TCK frequency requirements. 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 Periodic Interrupt Timer with Real-Time Interrupt 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). Chapter Appendix: Registers Under Protection Deleted this appendix (relevant content is now represented by the “Register Protection” chapter).
12-Oct-2012	3	Editorial change. Changed “LINFlex” to “LINFlexD”. Chapter Introduction Updated the dedicated number of channels for 12-bit ADC in family comparison table. Changed “Two MSEO (Message start/end out) pins" with "One MSEO (Message start/end out) pin" in Section 2.4.29, Nexus Development Interface (NDI) Updated Section 2.4.20, Serial communication interface (LINFlexD) Chapter Signal Description Revised the pinout information for the 176-pin QFP. Deleted “Remove MSEO1 port pin line” in Table 11, Nexus 3+ pin descriptions Replace MSEO[1:0] with MSEO in Note 8 in Table 10, Functional port pin descriptions and Table 13, Functional port pin descriptions. Updated the entries for PL[11] and PL[9] in Table 10, Functional port pin descriptions and Table 13, Functional port pin descriptions.

Table 813. Document revision history

Date	Revision	Changes
12-Oct-2012	3 (cont.)	Chapter Microcontroller Boot Updated the Figure 8, Boot sector structure and Section 5.1.1 Flash memory boot. Chapter Clock Description Added Note after CMU_MDR register. Added Note in Section , Crystal clock monitor: Note: Functional FXOSC monitoring can only be guaranteed when the FXOSC frequency is greater than (FIRC / 2RCDIV) + 0.5 MHz. Added Note in Section , FMPLL clock monitor: Note: Functional FMPLL monitoring can only be guaranteed when the FMPLL frequency is greater than (FIRC / 4) + 0.5 MHz. Chapter Clock Generation Module In the CGM_AC0_SC[SELCTL] field description, updated the auxiliary clocks for 0x0 and 0x1. Updated Section , Auxiliary Clock Dividers Updated the Figure 62, MC_CGM Auxiliary Clock 1 Generation Overview Chapter Mode Entry Module Added the clock sources in Section , Clock Sources (Main Voltage Regulator Independent) Switch-On and in Section , Clock Sources (Main Voltage Regulator Dependent) Switch-On. Chapter Reset Generation Module Made the necessary changes in NOTE after RGM_DES field description. Added Note in Section STANDBY Reset Sequence Register (RGM_STDBY). Updated BOOT_FROM_BKP_RAM field description in Table 101, STANDBY Reset Sequence Register (RGM_STDBY) field descriptions. Chapter Wakeup Unit Changed ‘WKUP’ to ‘WKPU’ all over the chapter to maintain consistency. Added CAN1RX in Port input function column of Table 114, Wakeup vector mapping. Chapter Real Time Clock / Autonomous Periodic Interrupt Added a note in APIVAL field in Table 127, RTCC register field descriptions Chapter CAN Sampler Updated “16 MHz fast internal RC oscillator” to “Divided 16 MHz fast internal RC oscillator” in Section 14.4 Functional description. Chapter Enhanced Direct Memory Access Updated the Section 17.6.8 Dynamic programming and its subsections. Chapter Interrupt Controller Updated the access mode of all the registers to supervisor mode only. Chapter Crossbar Switch Updated the name of column from ‘Logical Number’ to ‘XBAR port number’ and its content in the Table 185, XBAR memory map.

Table 813. Document revision history

Date	Revision	Changes
12-Oct-2012	3 (cont.)	Chapter Memory Protection Unit Updated Section 21.1.1 Overview making it compliant to MPU block diagram. Added note after Figure 200, MPU block diagram. Updated the EMN field of MPU_EDRn register. Updated footnote of Figure 206, MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2) Chapter System Integration Unit Lite Updated field description of MAXCNTx in Section , Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23). Chapter Inter-Integrated Circuit Bus Controller Module Updated the Section Generation of STOP. Chapter LIN Controller Added Section , Overrun in Master Mode Replaced “setting” with “resetting” in Section , LIN timeout mode Replaced “resetting” with “setting” in Section , Output compare mode Updated Section , Overrun and Section , Overrun Added a Note after Section 26.10.2, LIN interrupt enable register (LINIER) Deleted IFER[FACT] configuration table Updated the field description of FACT in Table 274 (IFER field descriptions) Chapter FlexCAN Updated the table title from “CAN Standard Compliant Bit Time Segment Settings" to “Bosch CAN 2.0B standard compliant bit time segment settings”. Modified the Note in Section , Protocol timing. Removed Abort Feature from MCR register. Changed the 19 bit of MCR to value ‘0’. Added a Note in RTR field description of Table 299, Message Buffer Structure field description. Reworded BCC field description in Section Module Configuration Register (MCR). Deleted text “For MCUs supporting individual masks per MB” from Section Rx Global Mask (RXGMASK), Section Rx 14 Mask (RX14MASK) and Section Rx 15 Mask (RX15MASK). Deleted “individual Rx mask Per Message buffer" note in the Section Rx Individual Mask Registers (RXIMR0–RXIMR63). Deleted "Matching Process" section (within Section 27.5 Functional description). Deleted Abort Feature from the chapter. Removed the Note after Section 27.2.2 FlexCAN module features. Removed the Note in CLK_SRC field from Section Control Register (CTRL). Removed the Note in Section Protocol timing that states about clock selection may not be available if the MCU doesn't have a PLL. Deleted WAK_MSK and WAK_SRC fields in MCR register and all instances that refer to wake up interrupt in other sections of this chapter. Chapter Fast Ethernet Controller Updated the fields names in Figure 575,Receive Descriptor Active Register (RDAR) and Figure 576, Transmit Descriptor Active Register (TDAR)

Table 813. Document revision history

Date	Revision	Changes
12-Oct-2012	3 (cont.)	Chapter Timers Added Note regarding Input frequency limitation related to IPM mode in Section , Input Period Measurement (IPM) mode. Chapter Analog to Digital Converter Updated the Presampling Voltage from Vss_HV_ADC1 to VDD_HV_ADC1 when 01 is selected as PREVAL value in Table 611, Presampling voltage selection based on PREVALx fields. Added a footnote in Figure 663, Implementation of ADC_0 and ADC_1. Added footnote in Section 32.1.1, Device-specific pin configuration features. Updated the field description of DSD in DSDR to “DSD x1/frequency of ADC clock. Added footnote to Table 559, ADC channel mapping. Added the description of Wait State of ADC.STATUS field as note in Table 564, Main Status Register (MSR) field descriptions. Updated Section Normal conversion operating modes Added Note in Section Abort conversion Updated Section CTU in trigger mode. Added footnote to EOCTU, JEOC and EOC fields of Table 565, Interrupt Status Register (ISR) field descriptions Changed the write bits of ISR, WTISR_ADC_0, WTISR_ADC_1, AWORR0 (ADC_ 0 and ADC_1), AWORR1- ADC_0, AWORR1- ADC_1, and AWORR2- ADC_1 to w1c. Added Note in Section , CTU in trigger mode Chapter Cross Triggering Unit Updated the description of CLR_FLAG in CTU Event configuration register. Added the NOTE at the end Section 33.4.1, Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63) Chapter Flash Memory Updated the heading of section from ‘Data flash memory memory’ to ‘Data flash memory Added Note in PFCR0 register. Updated Table 653, Programming NVUSRO_1 and STCU fault grading parameters. Updated the B1_RWSC field of PFCR1. Removed WWSC field from PFCR0 and PFCR1. Added Section 35.2.6, STCU programming using Flash. Chapter Error Correction Status Module Updated the field description of RAM_WS in Table 701 (MUDCR field descriptions) Chapter Self-Test Control Unit Updated the address offset of STCU MBIST Control Register to 0x0300 + ((k-1) × 0x4). Updated Section 39.5.4, Self-Test sequence after reset trigger. Chapter Cryptographic Services Engine Added Note after the Table 752, LOAD_KEY Command in Section 40.5.7 Load Key. Added Note in Section 40.5.19 Debug Authorization. Chapter JTAG Controller Added the list of TAP codes in Table 768, JTAG Instructions.

Table 813. Document revision history

Date	Revision	Changes
12-Oct-2012	3 (cont.)	Chapter Nexus Development Interface Added another Note in Section , Programmable MCKO frequency stating about MCKO_DIV value when Z0:Z4 frequency ratio is 1:2. Removed “System clock locked status indication via MDO[0] following power-on reset” feature from Section 42.5.2, NPC features. Added Section , Multi mode Nexus Tap with figure. Added missing section NPC_HNDSHK module. Removed DDR_EN field from Port Configuration Register. Change "2 message start/end out (MSEO) pins" with "1 message start/end out (MSEO) pin" in Section 42.1, Introduction. Replace MSEO[1:0] with MSEO in Figure 852,NDI functional block diagram. Replace MSEO[1:0] with MSEO in Figure 853, NDI Implementation block diagram. Replace "Two MSEO (Message start/end out) pins" with "One MSEO (Message start/end out) pin" in Section 42.2.1, NDI Features. Replace MSEO[1:0] with MSEO in Table 769, Signal Properties. Replace "The MSEO pins are used" with "The MSEO pin is used" in Section , Output Message Protocol. Remove MSEO1 in Figure 866, Nexus3 Functional Block Diagram. Replace MSEO[1:0] with MSEO in Section 42.6.4, Enabling Nexus3 Operation. Remove NOTE "No single MSEO mode with be implemented, only dual mode" in Section 42.9.1, Nexus3+ Auxiliary Port. Updated Figure 860, MSEO Transfers. (replaced 2-bit transfer by 1-bit transfer). Deleted NPC_HNDSHK module section as it was repeated.

Table 813. Document revision history

Date	Revision	Changes
11-Jan-2013	4	Chapter Throughout Deleted Preliminary Footer Chapter Voltage Regulators and Power Supplie Updated the reset value of VREG_PDMODE register to 0x0001_0000 Chapter Enhanced Direct Memory Access Added *Section 17.5.2 eDMA performance* Chapter System Integration Unit Lite Updated eMIOS to DSPI mapping in *Section 24.5.3.17 Parallel Input Select Register (PISR0—PISR15)* Chapter Analog-to-Digital Converter Updated sentence in *Section 32.4.3 ADC sampling and conversion timing* clarifying that it is only for 10-bit Chapter Flash Memory Updated the field description of PAD3V5V[0] and PAD3V5V[1] in Nonvolatile User Options register Moved CSE_RUN_MODE field to bit 31 of NVUSRO_1 register Chapter Timers Updated the reset value of MDIS and MDI_RTI field in field description in *Section 31.4.5.2: PIT_RTI Module Control Register (PITMCR)* from 0 to 1 Chapter Self-Test Control Unit Updated the reset values of the registers. Updated the reset values in *Table 721: STCU registers configuration after the boot sequence phase 1*.
18-Sep-2013	5	Updated Disclaimer.

Table 813. Document revision history

Date	Revision	Changes
19-Jan-2015	6	Chapter Clock Description In *Figure 30: SPC564Bxx and SPC56ECxx system clock generation*: – Added a footnote – Updated the clock configuration for FMPLL. Chapter Clock Description Module (MC_CGM) Updated register description section of *Section 7.4.1.1 e200z0h Core Clock Divider Configuration Register (CGM_Z0_DCR)*, *Section 7.4.1.2 FEC Clock Divider Configuration Register (CGM_FEC_DCR)*, *Section 7.4.1.3 Flash Clock Divider Configuration Register (CGM_Flash_DCR)*, *Section 7.4.1.7 System Clock Divider 0 Configuration Register (CGM_SC_DC0)*, *Section 7.4.1.8 System Clock Divider 1 Configuration Register (CGM_SC_DC1)*, and *Section 7.4.1.9 System Clock Divider 2 Configuration Register (CGM_SC_DC2)*. Chapter Mode Entry Module (MC_ME) Added a footnote for STOP mode in *Table 75: MC_ME mode descriptions*. Chapter Reset Generation Module (MC_RGM) Added a note in *Section 9.3.1.5: Functional Event Alternate Request Register (RGM_FEAR)*. Chapter Power Control Unit (MC_PCU) In *Table 106: MC_PCU Memory Map* changed address of PCU_PCONF3 from 0xC3FE_800 to 0xC3FE_800C. In *Table 109: PCONF settings for RAM in STANDBY* changed size of STANDBY RAM from 60 (8 + 56) to 64 (8 + 56). In *Figure 104: Power Domain #2…3 Configuration Registers (PCU_PCONF2…3)* changed Address from 0xC3FE_800 to 0xC3FE_800C. Chapter Voltage Regulators and Power Supplies In *Section 11.1: Voltage regulators* updated note and replaced PD3 and PD4 domains with PD2 and PD3 domains. Updated the bit description of 5V_LVD_MASK bit in *Table 112: VREG_CTL field descriptions*. In *Section 11.3: Power domain organization* replaced the sentence “two primary power domains” with “four primary power domains” Updated *Figure 107: MC_PCU Events During Power Sequences (STANDBY mode)* Chapter Wakeup Unit (WKPU) Updated Note below *Table 124: WIPUER field descriptions*. Chapter e200z0h Core: In *Section 15.2: Features* removed Testability feature description. Chapter Enhanced Direct Memory Access Channel Multiplexer (DMA_MUX) In *Table 165: DMA_MUX memory map*, changed the last entry from 0x31 to 0x1F.

Table 813. Document revision history

Date	Revision	Changes
19-Jan-2015	6 (cont.)	Chapter System Integration Unit Lite In *Table 225: Peripheral input pin selection*, PADSEL13 field's functional / peripheral entry E1UC[3] / eMIOS 0 changed to E0UC[3] / eMIOS 0. Chapter LIN Controller (LINFlexD) Added a note in *Section 26.10: Memory map and register description*. Added a figure note in *Figure 343: Programming consideration: response timeout(1)*. Added footnote for DRIE and DTIE bits of LINIER register in *Table 261: LINIER field descriptions*. Updated IOT bit of LINTCSR register in *Table 266: LINTCSR field descriptions*. Chapter FlexRay Communication Controller Updated the title of *Figure 570: Transmit Data Available*. Chapter Flash Memory Added new rows and removed the NVUSRO_1 entry from *Table 626: Shadow sector structure*. In *Table 656: NVUSRO field descriptions* updated description of PAD3V5V[0] and PAD3V5V[1] fields. Chapter Cryptographic Services Engine (CSE): In *Table 744: CSE_SR field descriptions* updated description of BFN field. Updated *Section 40.4.6: Secure Boot*.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved